PHILIPS

Philips Medical Systems Nederland BV

University of Twente
Enschede - The Netherlands

REPORT

DICOM Image Retrieval

Author
Gerald van Veldhuijsen

Date
28 August 2006

Tutors:
Gerard van Ballegooyen
Paul van der Vet
Theo Huibers

Table of Contents

PART I

1 Introduction

1.1 Philips Medical Systems

Philips Medical Systems (PMS) in Best is part of PMS global. PMS produces medical equipment. The business activities of PMS are divided in three categories: Imaging systems, customer services and clinical solutions.
Imaging systems consist of medical equipment, used to create images of various parts of the human body in varied detail for radiologists and cardiologists. Examples of this equipment are X-ray machines and CT-scanners.
Customer services include consultancy, which consists of analysis and advice, and the financing, maintenance and repair of medical equipment.
Clinical solutions include healthcare IT systems as well as patient monitoring and cardiac devices. Examples of healthcare IT systems are archiving systems for medical images, viewing stations and medical printers. Examples of the second category are the automated external defibrillators.

The IT department of PMS is responsible for the software of the medical equipment. Our project has taken place in the Interoperability department of PMS. The Interoperability department is a testing department of the IT department. A great deal of the software of the medical equipment must operate according to the DICOM (Digital Imaging and Communication in Medicine) standard. The Interoperability department tests whether the software characteristics are conform the standard. These tests consist of two types. The first type of test checks whether a system on its own operates conform the standard. The second type of test has the purpose to check whether the system is able to communicate with other medical devices conform the standard.
Of course the level of testing is very high, because the subject of testing is medical equipment. The correct functioning must be guaranteed before a system can be approved and taken into production. Medical (DICOM) images play a very prominent role in these tests.

1.2 Assignment

This section contains the original assignment description that served as the starting point for our project.

With the DICOM standard an enormous set of medical images can be exchanged. A DICOM image contains the actual image data together with several descriptive information blocks. Within the Interoperability department there are many CD's available containing different medical images. An important part of the process of approving a product is the validation of the DICOM interface. For this validation, these medical images are used.

Depending on the system to be tested a selection has to be made from the large amount of images available. All available images have to be searched through to see whether the image data and descriptive blocks conform the conditions of the system to be tested. As the number of available images is increasing enormously (the number of CD's is already in the order of hundred's) it is almost impossible to do this job by hand. As an extra difficulty, the CD's are not all stored in a central place, but are distributed over the department.
Within Philips Medical Systems there are a few systems available that are able to store images and to retrieve them. It is however not possible to search these systems on conditions relevant for testing. Within the Interoperability department tools are available to extract the relevant data from the medical images. The main subject of the assignment was to find a flexible solution to this search problem.

A DICOM retrieval application had to be developed that can use the different available tools. With this application it should be easy to search for relevant images and retrieve these. In the process of development, first of all requirements had to be defined in a User Requirements Specification. Based on these requirements an inventory had to be made of the current systems and tools available. After that a proposal had to be done of the new application. After approval this application had to be implemented as well as tested. Communication with colleagues about the requirements and the functional use of the application played an important role.

1.3 Document Organization

This document will describe our project that has dealt with the above-described assignment. The whole document is split up in 5 large parts, i.e. Problem Definition, Method, Implementation, Testing and Conclusion.
The rest of this document is organized as follows. After this introduction Part I continues with a section (2) on the DICOM standard. As the DICOM standard is the backbone of the images that we are dealing with, it is an important part of our project. In this section we introduce the standard, we describe the relevant basics of the standard and we will try to depict the complexity of the DICOM standard. Next we deal with the requirements specification of our system in section 3. Then we continue with section 4 that discusses the related work in the domain of our problem.
In Part II we start with section 5 on data retrieval versus information retrieval. Here we will explain the differences and resemblances between the two concepts as well as what aspects we need of both. We continue with DICOM XML in section 6. The first step in our approach to solve the problem was to use XML as the document format for our data collection. Here we explain why we chose this approach and what the characteristics of this approach are. After this, in section 7 we deal with the Terrier platform that we have used as the basis for our system. We describe the important characteristics of Terrier here. Then in section 9 we describe the global approach of our total system. Here we introduce the division between the implementation that is a modification of the Terrier platform and the other implemented parts.
In the next two sections in Part III we then continue to go into the implementation in more detail. In section 10 we deal with the extensions and modifications of the Terrier platform. In subsection 10.1 we describe the new index process along with the different index structures. In subsection 10.2 we deal with the weighting mechanism of our system and in 10.3 we discuss the performance optimizations we had to make to the Terrier platform. In section 11 then we describe the implementation that is not related to the Terrier platform. In this section we deal with the input side of our system, i.e. how the data enters the system and how it is processed, as well as with the output side. In the description of the output side we deal with the way queries are used and with the presentation of the results.
After we have dealt with the implementation of the system, in Part IV, i.e. section 12, we describe the experiments we did to verify the functionality of the system. We conclude our report in section 13, which is the content of Part V, with a discussion of the results and an outlook to future work.

2 DICOM Standard

The Digital Imaging and Communications in Medicine (DICOM) standard was originally published as the ACR-NEMA standard. This was the result of the founding of a joint committee between the American College for Radiology and the National Electrical Manufacturers Association that had the purpose to develop interfaces and standards relating to imaging equipment and other medical electronic equipment. [1]
After that the DICOM standard evolved to its current version, DICOM 3.0. It is now maintained by a special DICOM Standards Committee and is published by the NEMA [2]. The current version is called 3.0 to show that it was based on the ACR-NEMA standard version 1.0 and 2.0. Although the name and version remains DICOM 3.0 an updated edition of the standard is published each year. In practice this means that the standard grows with each update. Nowadays the standard has been adopted by virtual all suppliers of imaging products.

The DICOM standard can be seen as a set of engineering information that describes the information structures and procedures that control the input and output of data from medical imaging systems [3]. The standard describes how medical images and its associated information should be formatted and how to exchange them. Note that the term ‘image’ can be misleading here, because DICOM objects do not necessarily need to contain an image at all. An example of this is a structured report (SR). Like the name suggests this is a structured report about a medical investigation of a patient. It can link to images, but does not contain an image itself. We will use the terms DICOM image and DICOM objects interchangeable throughout the rest of this document.
DICOM Interfaces are available for connection of any of the following categories of digital imaging devices: (a) image acquisition equipment; (b) image archives; (c) images processing devices and image display workstations; (d) hard-copy output devices (printers) [3]; (e) information systems (also called scheduling systems). By means of the standard it is possible for all these types of devices to

communicate with each other. For this the DICOM standard contains extensive descriptions of protocols, services and information objects that make this communication possible.
The DICOM standard changes continuously over years. Every now and then new medical imaging data fields and higher-level information objects are introduced to support new techniques (like 3D-imaging recently). On the other hand, some parts are phased out because the corresponding techniques are not used anymore.

The two fundamental components of DICOM are the information object and the service class. Information objects define the core contents of medical imaging, and service classes define what to do with those contents [3]. A distinction is made between service class users (SCU) and service class providers (SCP). For example DICOM printing is a service class. The actual DICOM printer is the SCP and a workstation that is capable of sending the appropriate messages in order to let it print would be considered a DICOM print SCU.
At the implementation level, DICOM specifies a file format for storing data. The part of the file format that contains the descriptive information is called the DICOM header. This is for us the most interesting part of the standard, because our problem deals with a collection of stored DICOM data and not with the communication between different DICOM devices. We must note here however that this header is not very different for stored objects and objects that are the subject of communication.
The most important feature that distinguishes the DICOM header from the many other image standards is that it is not restricted to image data alone. Also image related information is stored and like we mentioned the image data even is not necessarily always present.
The specification of the DICOM header is described almost entirely in parts 3, 5 and 6 of the standard. The basic building blocks are the DICOM attributes. Such an attribute consists of a tag, a value representation (VR) and a value field.
Part 6 contains all tags that can exist in DICOM data. A tag consists of two 16-bit numbers. Because they are derived from the old ACR-NEMA standard is the first part known as ‘Group’ and the second as ‘Element’ [4] . Together they form the attribute tag and they are usually shown in parentheses. For example the attribute tag of a patient name is (0010,0010) and the study description has (0008,1030) as tag. The number of different tags defined in the standard lies in the order of thousands. The attribute tag is a unique identifier. This means that although the standard can change, this identifier will always belong to the same attribute.
Part 5 describes the different value representations and their value ranges. For example attributes that have Patient Name (PN) as the VR have a maximum length of 64 characters. This is all specified in part 5.
Part 3 lists the different information objects recognized by the DICOM standard. It describes what is contained in an information object; it specifies which attributes must be present and which are optional. As an example “Digital intra-oral x-ray image” is an information object, where the attributes “Patient’s Name” and “Pixel Data” are two of the mandatory attributes.
The DICOM standard not only specifies the DICOM header itself, but it also specifies how this data should be stored on removable media such as diskettes, optical disks, CD’s and DVD’s. DICOM specifies the DICOM header and a file directory. This file directory is called DICOMDIR. Through this DICOMDIR the other DICOM images stored on the medium can easily accessed.

The DICOM standard is not always very straightforward to use and understand. This is not only due to its extensive size. First of all can information objects can be used recursively, possibly resulting in complex nested structures. Further, besides the classes and attributes defined in the standard, users are free to use self-defined attributes, so called private attributes. Finally, the standard is ambiguous and (partly as a consequence) it gives the possibility to create redundant information. It is ambiguous because the same data can be represented in multiple ways and redundant because the same data can be stored in different locations. Different locations can be the different information objects within a DICOM object, but also different DICOM objects itself.
One cause of this seeming complexity is the way the standard is built up. As we mentioned previously, a special DICOM Standards Committee maintains the standard. Actually the current standard is the result of the joined effort of all big vendors of medical equipment and all these vendors have members in the committee. A lot of parts of the standard are based on compromises between different vendors. A proposal that in itself would be an improvement, for example in the field of ambiguity, but also would be in favor of only one (a few) of the vendors, will probably not pass the counsel.

At the Interoperability department the DICOM standard is mainly used in the context of conformance statements. Vendors of medical equipment are required by the Standard to provide conformance

statements that accurately describe their DICOM implementation. Due to the complexity of the standard no products currently implement it totally. A conformance statement is a document that clearly describes what parts of the DICOM standard are supported by a particular piece of equipment. The main purpose of a conformance statement is to easily give a verdict on whether system A is capable of correctly communicating with system B. The Interoperability department completely tests the functionality of a system and can therefore check or generate a conformance statement.
[5] [2] [3] [1] [4]

3 Requirements

This section describes the functional requirements of the system. In Appendix A the complete functional requirements document can be read. The testers of the Interoperability department are the main group of potential users. The images in the Interoperability department are not used for testing purposes only. They are also used for promotion and demonstration purposes. Therefore the application is not restricted to the group of testers only. Other users that should be able to use the system could be people working at the marketing department who also often search for suitable images for this reasons. The other way is also true, i.e. that images that were used for demonstrations in the first phase are also be used for testing purposes afterwards.

The activities of a tester in the Interoperability department related to the search for images are the following. Based on the characteristics of the system to test (=Conformance Statement) the images that cover the required functionality of the system under testing must be selected. The DICOM retrieval system that we have developed should help the user in finding the most suitable images. Sometimes it is possible to use an artificial image. There are tools that can construct an artificial image. They take an image and some descriptive data and put this together into DICOM format. Besides the fact that this takes more time then using existing images, it is most of the time preferred to take images from the real world. The preference to use real world examples is a common phenomenon in the test domain.
The support for the user to easy search for images puts requirements on the search and retrieval capabilities of the system as well as on the presentation of the results. We discuss these two types of functional requirements separately. Before that we shortly deal with the requirements of the storage of images into the system and after it we discuss the requirements for the system that are implied by the explicit functional requirements.

The basis of the system is a collection of data to search in, a collection of DICOM images. To create such a collection users must be able to load this data into the system. The system must be able to accept all DICOM objects that are available. The images can be offered to the system as single images or as sets of images. In the latter case this set of images is provided with a DICOMDIR as is described in previous section.
During the process of loading images into the system, the user must be able to add his own description to the data he is providing. In the rest of this document we will refer to this a user-defined meta-data or just meta-data. In the case of a single image this meta-data belongs to that image only, but in case a set of images is uploaded the meta-data is applicable for the whole set. The minimum content of the meta-data is the CD-description of which the images originate from. In the original situation the description on the CD-case was the main identifier in the search process. This information must be maintained. The user-defined meta-data can vary from only this CD-description to a short description of the test for which it was used. There is no limit to the length of this data.
One type of information that is part of this user-defined meta-data must be defined explicitly. This information tells whether the specific image(s) are clinical or non-clinical. As we mentioned previously, there is a preference to use real world examples for testing purposes and this information plays an important role in this selection.

Clinical vs. Non-Clinical
Clinical data contains DICOM objects that originate from medical systems in hospitals. This means that these objects contain real body parts. Non-clinical data is generated purely for testing purposes and can contain everything, for example a stapler. Clinical data that is used is (should be) anonymized, so that the privacy of patients is guaranteed.

Next we deal with the retrieval requirements. These requirements are based on the main idea that users don’t need complete knowledge of the structure of the DICOM images. However when they do have this knowledge they must be able to use this knowledge to improve the retrieval results. This

idea results from the fact that the users of our system vary in their expertise level with respect to the DICOM standard. Further like we indicated in the previous section is the DICOM standard simply too extensive and too variable to have complete knowledge of.
Based on this idea we can distinguish the different types of querying that must be supported. First of all it must be possible to query on content only. With content only we mean queries that consist of simple keywords; this is comparable with the way current information retrieval systems are used, like Google.
The matching of the keywords in the queries with the images in collection should be based on fuzzy matching. Here again we can make the analogy with Google that will also return results that contain terms that almost but not exactly match the keywords from the query.
Further we need to support queries where content and structure are combined. These are so-called containment queries where is searched for keywords that must occur in the context of specific structural parts of the DICOM standard. The lowest level of this structure is the level of the DICOM attributes. So for example it must be possible to query for keywords in a specific attribute.
Finally we have another two types of queries that must be supported: range queries and exact matching queries. These types of queries are very useful when the boundaries of a system are tested because this way the images can be retrieved with characteristics that are on those specific boundaries. The structural part plays an important role in this type of queries. The queries consist of a structural part and value together with one of the operators '=' or '<'.
Another retrieval requirement is that the meta-data that was added during the storage process must be taken into account in the retrieval process. This means that when information contained in the user-defined meta-data somehow matches a query, the images that belong to this meta-description must be put into the result set. We are not interested in retrieving the meta-information itself, but it should cause its related images to be returned as result.
Finally, all information contained in the DICOM objects must be available for querying. This is to be prepared for changes in information needs of the users. At this moment some information of the DICOM objects may be very relevant to base the selection of images on, but due to various reasons other information can become relevant in the future as well or instead. When simply all information is available for querying this process will not lead to problems.

For the presentation of the results the most important requirement is that not necessarily all available results for a query have to be returned, but when there are result available at least some must be returned. Further the returned results should be ordered according to relevance. In this ordering it is not necessary that the best result will be listed first but at least the worst result should not be listed first. Next we have to answer the question what is determinative for DICOM object to make it more relevant then others. The answer to this question can be divided into two parts.

- Documents can be ranked with respect to the relevance of the keywords
- Documents can be ranked with respect to the relevance of the structural information

There are some aspects that play a role in the relevance of a document with respect to the keywords of the query. First of all a document that contains for example the word 'mr' does not necessarily has to be a DICOM image that originates from a mr (Magnetic Resonance) system. Most probably this is however the type of image the user is looking for in that case. So the application must use other techniques than simply return the documents that contain the requested keyword.
Further when multiple keywords are used in a query, all this keywords can contribute to the returned result set of images. Some keywords however will lead to more relevant result than others. For example the keyword 'image' occurs in nearly every DICOM image. This keyword is then less likely to contribute to relevant results compared to keywords that occur less frequently. Also with this phenomenon the application has to deal with when retrieving images.
Some parts of the document structure are regarded to contain more relevant information than others. Currently, these are typically the fields that are used in the current available systems for querying. When information that is searched for occurs in any of these parts, it should be possible to rank these documents higher. In other words, it should be possible to assign a higher weight to some structural parts and it this must be flexible because it can change over time.

Besides the ordering of the results it is important how the different results are presented to the user. Of each returned DICOM image a preview must be shown. This can help a user in determining the relevance of a returned image. Finally it must be possible that in a list of returned results the images that originate from the same set (during storage) are grouped together.

Finally we discuss the requirements that don't fit into any of the previous categories and/or are implications of the previous discussed requirements. In the discussion on the storage requirements we mentioned that it must be able to load all available DICOM objects into the system. In the previous section on the DICOM standard we mentioned that the DICOM standard is subject to change. The application must be flexible enough to deal with these changes in the future. This means no implementation work needs to be done to make sure that new DICOM images can be loaded into the system.
Last of all, the system needs to meet some performance criteria. The response time of the system has to be within certain limits, where the size of our data collection will be large. The exact size of our collection of images is not representative, because it consists of both image data and descriptive data. When retrieving we will only search through the descriptive data. The experiments must show the size of this descriptive data.
However we think of a response time in the order of seconds, where a response of more than ten seconds is undesirable. We take into account that we are introducing a new system to the users. Real reference material is not available, but it is at least important to make new users enthusiastic and a long response time does not contribute to that.

Summarizing all the discussed requirements we can say that we need flexibility for both input of DICOM object as for the information that can be queried. We need to be able to query both content as structure and in the retrieval process we take user-defined meta-data into account. Finally we need to order the returned results according to relevance.

4 Related Work

Our project is not the first attempt to solve the previous described DICOM retrieval problem. Within PMS there have been previous attempts to come to a solution. These projects are however not properly documented. The solutions vary from database-based solutions to solutions based only on Excel sheets that contain an inventory of all available images. Some of them were more successful than others but none of them led to constructive satisfying solution. The reasons for these other attempts to fail were basically the same for all these projects and can be divided into two causes:

- Changes in the structure of the information to be stored
- Changes in the information needs of the users

Like we described in the section 2 the DICOM standard can change and therefore the way the information is structured in the DICOM images changes and new types of DICOM objects can be introduced. Most of the previous systems were based on a certain (fixed) data model and therefore these new structures could not be added. To change the model and system mostly took too much effort and then still there is no guarantee for the future.
The second cause is related to the previous. The information that users of the systems are interested in changes over time. The problem with the previous systems was that they provided a predefined set of searchable information that was a subset of the total information available. Information that was outside the scope of these sets was not searchable. The greater importance this new information gained, the less useful the search systems became.

Picture Archiving and Communication Systems (PACS) are systems that are developed both outside PMS as well as inside as a business activity of PMS. They are mainly used in hospitals and these systems are capable of storing an enormous amount of images that of course also can be retrieved. These systems also always implement a DICOM interface and therefore it is possible for medical imaging equipment to directly send their images to a PACS. These PACSs are developed for the use in hospitals. The set of searchable information is rather small and is specifically chosen so that medical specialists can retrieve the necessary images, based on for example only the patient name. This lack of query possibilities already makes them unsuitable to solve our problem.
There have been projects where these systems serve as the basis for another retrieval system. This is done for example in [6] where a content based image retrieval (CBIR) system is setup with a PACS as the basic storing device. The actual CBIR engine has to implement a DICOM interface in order to be able to communicate with the PACS. At this moment it was not beneficial for to use such a PACS. First of all these are expensive machines and further would it have taken too much time to implement a correct DICOM interface. These arguments had more weight than the advantages of a PACS of

being a mature and stable storing device for medical images. Also does this not really make a difference to our proof of concept for our system, because our actual system would be a retrieval engine on top of a PACS.
In future work we can look at what the possible advantages of using a PACS are and how to integrate it into our system.

Outside PMS not much attempts can be found that tried to solve the problem. On of the few is the plug-in for the desktop variant of the Google search engine [7]. This plug-in simply processes the descriptive blocks of the DICOM images as flat files and indexes these. So this plug-in contains the intelligence of how it must decode the information in a DICOM object in order to process it. This project will not suffer from the two problems that we described that were problematic for the projects within PMS. It does not use a fixed model for the DICOM objects, so changes in the standard are not a problem, only if the way the information must be decoded changes. Further it does not use a subset of the information, but it offers all information that it encounters.
However it is not possible to use all the types of queries we defined in the requirements. The reason for this is that the structure of the DICOM images is ignored. Further it is not possible to use the user-defined meta-data we described and it is not possible to show a preview of the images in the results. Finally it is not an open source project so it is not possible to make changes based on the requirements of our project.

Finally there are some projects that provide more retrieval possibilities then current DICOM databases, but these systems are restricted to a specific domain like [8] and [9] that are restricted to only X-ray images and radiology images respectively. Because of these restrictions these systems don’t provide enough flexibility, as we want to have the flexibility to store all types of DICOM objects.

PART II

5 Data retrieval vs. Information retrieval

This section is meant to describe the relationship between data retrieval and information retrieval. The reason for this discussion is that in the description of the requirements in section 3 aspects of both concepts come to the fore. So here we first introduce both concepts, then we deal with the differences between the two. Then we look at which aspects of both concepts apply to our project and finally we look at the consequences.

The standard work that is currently referenced to when dealing with the concept of information retrieval is [10]. In this book the following definition of information retrieval is given:
Information retrieval (IR) is a part of computer science that studies the retrieval of information (not data) from a collection of written documents. The retrieved documents aim at satisfying a *user information need* usually expressed in natural language [10]. Today's IR-systems allow the user to enter keywords that identify the subject of which the user wants to find information. Such an IR-system then returns a list of results that are ordered according to the relevance of the documents. This ordering is based on a certain model and it is of course system specific.

In [10] also a definition of data retrieval is given: data retrieval (DR) is the retrieval of items (tuples, objects, Web pages, documents) whose contents satisfy the conditions specified in a (regular expression like) user query. The most well-known systems that support DR are SQL-based database systems. Both the data as the queries for these systems are well-structured.

The differences between IR and DR are clearly depicted by [11]. An overview of the differences is given in figure 1.

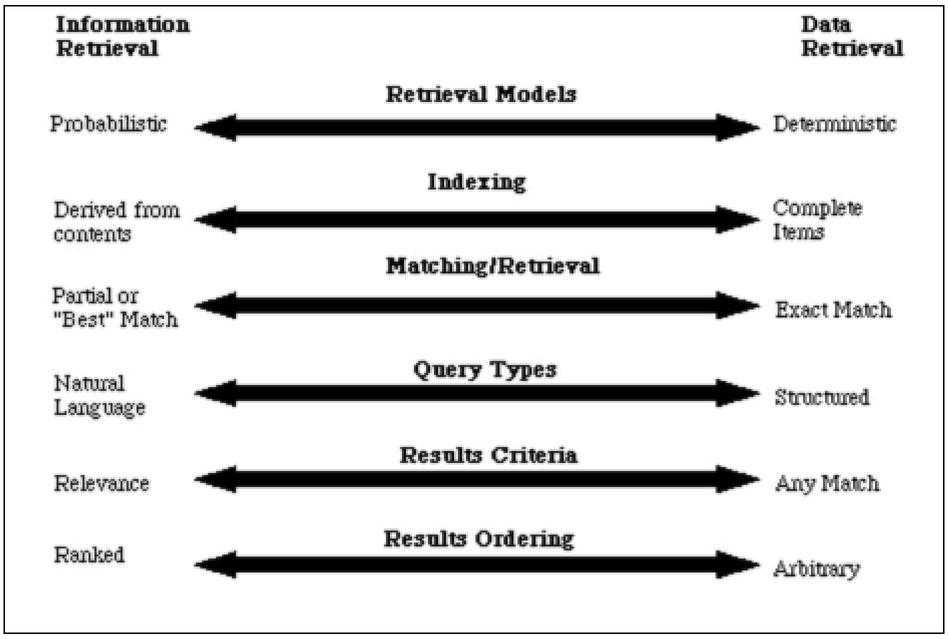


Figure 1. Information Retrieval vs Data Retrieval [12]

Next we deal with the question of why we would need both concepts. We relate this discussion to the requirements we discussed in section 3. We start with the aspects of the requirements that are part of the domain of IR. First of all we want to support fuzzy matching in our retrieval process, in figure 1 this is described as 'partial or best match'. The most important technique that is used to support this is stemming. Stemming is a well-known technique in the IR domain that normalizes terms to a base form before matching.

Further in the requirements we discussed that we wanted the results to be presented in a ordered way, i.e. ranked according to relevance. As we can see in figure 1 the ranking of results according to relevance is part of IR.
Finally we discussed the requirements that we needed flexibility with respect to both the document structure at the input side as the information availability at the output side. This is not a specific characteristic of IR in general, but a large part of current IR systems has this characteristic. We already saw in section 4 on the related work that the DICOM plug-in for the Google desktop meets the requirement that it does not put requirements on the structure of the documents and also that it has all information that is contained in a DICOM object available to search through.

In figure 1 we see that for the query types and retrieval for DR holds that the queries are well structured and results are only delivered in case of exact matching. This corresponds to our requirements that we must support the query types exact matching and range matching. The queries in these cases are structured, because they always consist of a field in combination with a value(s). Retrieval is based on exact matching because documents either satisfy the condition posed in the query or not. This corresponds with the definition of DR.

The main challenge for our DICOM retrieval system was to combine these two types of retrieval into one fully functional system. That means that we want to exploit the use of structure from data retrieval while still maintaining the flexibility of the information retrieval concept.

6 DICOM & XML

This section describes the first step that we took in our approach to solve the problem. Instead of directly using the images in DICOM format we have used XML representations of the descriptive data of the DICOM images. First we discuss why we have chosen to work with XML as an intermediate format and after that we describe how the XML format is used and what the effects of this approach are.

Original DICOM data is in binary format. In principle this binary DICOM stream can be used to transfer data inside and between programs (and it is of course). However this would require each program (component) to be aware of the fine nuances of how to decode and encode the binary DICOM stream. Therefore an XML representation of the DICOM data is a useful alternative [13]. The author of [13] as well as other experts suggest that an XML representation should be become part of the DICOM standard but up till now no real results in that direction have been achieved. Most of the current research focuses on the structured reports (SR) instead of the DICOM standard as a whole.
A DICOM image is like an XML document hierarchically structured. Therefore using XML as the representation of DICOM images is a logical choice. There are a number of advantages of using XML compared to DICOM. XML is far more widely used than DICOM and XML's popularity is still increasing in various fields. Therefore a lot of tools and parsers are available for processing and manipulating the data. Furthermore a lot of research is done in the field of XML retrieval. By using XML to represent the DICOM data the main subject of our project, the retrieval of DICOM data, internally transforms to the subject of XML retrieval. Finally, as stated, the DICOM standard can be (and is) changed and extended. XML itself and XML tools are in principle very suitable for dealing with this phenomenon.

For creating XML representations of the DICOM images we make use of a validation tool for DICOM images, DVT [14]. This is an open source project, which is maintained by the Interoperability department of PMS.
Simply put, this toolkit validates the communication between medical equipment as well as the generated DICOM objects in order to check whether violations to the DICOM standard have occurred. When a DICOM image is validated, the output of the tool is an XML file that completely covers the information that is contained in the DICOM image together with additional validation information. This additional information for example contains the number of DICOM violations that were found in the specific DICOM image. It is this file that we use as the representation for our DICOM images.

```
  . . .

<Attribute Group="0018" Element="1151" Type="2C" VR="IS" Length="4"
        Present="+" Name="X-Ray Tube Current">
 <Values>
  <Value>603</Value>
 </Values>
</Attribute>

      . . .

<Attribute Group="0028" Element="0004" Type="1" VR="CS" Length="12"
                        Present = "+" Name="Photometric Interpretation">
 <Values>
  <Value>MONOCHROME2 </Value>
 </Values>
</Attribute>

      . . .

<Attribute Group="0008" Element="2111" Type="3" VR="ST" Length="60"
                  Present = "+" Name="Derivation Description">
 <Values>
  <Value>Compress BN JPEG Lossless, Decompress Pegasus JPEG Lossless
  </Value>
 </Values>
</Attribute>

      . . .

<Attribute Group="0040" Element="A160" Type="1C"
      VR="UT" Length="478" Present="+" Name="Text Value">
 <Values>
  <Value>Examination shows all the dorsal vertebral bodies are of normal stature and contour with
normal signal intensity. There is decreased signal in the disc at levels suggesting desiccation of the
disc. The remaining dorsal discs show normal signal intensity. The dorsal spinal canal shows normal
size and configuration with normal signal intensity. The dorsal spinal cord is not enlarged and shows
normal signal intensity. Paraspinal soft tissue shows normal signal intensity.
  </Value>
 </Values>
</Attribute>

      . . .

<ValidationCounters>
 <NrOfValidationErrors>0</NrOfValidationErrors>
 <NrOfValidationWarnings>0</NrOfValidationWarnings>
 <NrOfGeneralErrors>0</NrOfGeneralErrors>
 <NrOfGeneralWarnings>0</NrOfGeneralWarnings>
 <NrOfUserErrors>0</NrOfUserErrors>
 <NrOfUserWarnings>0</NrOfUserWarnings>
 <ValidationTest>PASSED</ValidationTest>
</ValidationCounters>

   . . .
```

Figure 2: Example of (part of an) XML representation of DICOM object

In figure 2 an example of a part of how such an XML representation of a DICOM object could look like is given, generated by DVT. It shows both information that originates from the original DICOM object as well as added validation information.
Important characteristics of a DICOM XML document that do not follow from the given example are the following. The first is a consequence of the characteristics of the DICOM format. Attributes can contain a sequence of other attributes. Attributes in such a sequence again can contain a sequence of attributes, etc. The effect is that the DICOM XML document can contain nested structures.

The next characteristic is also a direct consequence of the DICOM format. The same information can be placed in several attributes. The effect is that it is not (always) clear in which attribute the useful information can be found.
The last characteristic is a result of the used XML generation tool, DVT. Besides the summary of validation results also validation information is shown at the location that it applies to. This means that throughout the document structures of error messages and warnings can occur.

This approach of using an external tool, DVT, to generate our XML representation has some consequences for the requirements of our system. In the requirements we stated that the structure of the DICOM objects is variable. Therefore also the structure of the XML documents can change. Besides this, not only will lead changes in the DICOM standard to changes in the XML document, but also changes in DVT will lead to changes in the XML structure. Because there is no standard that prescribes how an XML representation of a DICOM image looks like, we cannot put restrictions from our side either on the XML documents. Our goal was that our system is flexible enough with respect to the generated XML representation that it also accepts XML representations generated by other tools. In figure 3 an example of a XML representation generated by a tool other than DVT is given.

```
      . . .

<DerivationDescription element="2111" group="0008" vr="ST">
 <value number="1">Compress BN JPEG Lossless, Decompress Pegasus
  JPEG Lossless
 </value>
</DerivationDescription>

      . . .

<PhotometricInterpretation element="0004" group="0028" vr="CS">
 <value number="1">MONOCHROME2</value>
</PhotometricInterpretation>

      . . .

<XrayTubeCurrent element="1151" group="0018" vr="IS">
 <value number="1">603</value>
</XrayTubeCurrent>

  . . .
```

Figure 3: Example of fragment in a different representation

Further, in the requirements we have specified that users are not obliged to have complete knowledge of the underlying DICOM format in order to be able to work with the system. Now because we are actually dealing with XML retrieval this has slightly changed into the requirement that users are not obliged to have complete knowledge of the underlying XML structure.
Finally, we discussed the requirement that all information should be available for querying. Now when we use the XML documents as representation for the DICOM images we maintain this requirement. This means that the information that is added by the XML generator tool will also be available for querying. This will add useful functionality to the system, because for example the validation information can also be queried, which can be valuable information for testers.

This section can raise the question what basic knowledge a user must have in order to use the application. To basically query the collection the user must have knowledge of the domain specific terminology that can occur in the DICOM objects. This requirement is not different than for other retrieval systems. Further to use the structure in the queries the user must be aware of the names of the different structural parts. Most XML generator tools leave the name intact that was defined by the DICOM standard. To be able to query on features that are unique for a specific XML representation, then the user must have knowledge on how that specific XML structure is build up.
Later in this document in the part on the implementation some techniques will be discussed that will help the user to use structure in his queries.

User-defined Meta-information
In the requirements we described that the user must be able to add so-called user-defined meta-information at the time of storage. This information is also stored in an XML document. The advantage of using XML for this data is that we can treat both types of information the same way. For each dataset of images or single image such a meta XML document is created. In case of a dataset the information contained in this meta-file applies to all images in the dataset. Besides the types of meta-information that we described in section 3, all other information that users want to add will be placed in that file.
Because the meta-information must be taken into account at retrieval time, our approach makes sure that somehow their will be a link between the XML documents that are representations of the DICOM images and the XML documents that contain the user-defined meta-information. This mechanism will be described in section 10, where the retrieval mechanism is discussed.

XML generation tools
We use DVT as the tool to generate our DICOM XML representations for the images. Besides DVT there are several tools that are capable of generating XML representations. DCMTk [15] is an open source DICOM toolkit that implements large parts of the DICOM standard. Like DVT also DCMTk has the possibility of validating DICOM objects against the standard. However the module that provides the validation functionality is not free to use. Further, Dcm4che [16] is a open source DICOM implementation in Java, and one of its features is that it can translate DICOM objects to XML. Finally Pixelmed [17] is a DICOM implementation also in Java. Besides generating XML for DICOM images, it provides viewing functionality as well limited validation functionality. However this implementation is not as stable as the previous three.

7 Terrier

This section will introduce and discuss the Terrier information retrieval platform [18]. This is the system that we used as our base information retrieval system. Here we will discuss the global process of the platform.

Terrier is an open source project and it is meant to provide a platform for the rapid development of Web, intranet and desktop search engines. More generally, it is a modular platform for the rapid development of large-scale Information Retrieval applications, providing indexing and retrieval functionalities [19].
The decision for using Terrier as our base information retrieval system is based on a number of reasons. First of all the Terrier platform is an open source project so it is free to use, which was one of the requirements. Secondly, the Terrier platform is set up in a modular way, making it easier for us to implement our modifications and extensions to this platform. The system achieved good results on TREC 2004 and 2005 [18], so it can be regarded as a mature system. Finally, the Terrier platform was designed to work with an enormous amount of data, which is also the case for our project (The name Terrier stands for TERabyte RetIEveR, which indicates the purpose of the Terrier project).

We will continue by introducing the Terrier platform in general, where the aspects that are important for our application are covered in some more detail. For a more in depth description of Terrier we refer to the documentation of [19].
We can divide the functionality of Terrier into two parts, although they are directly related to each other: Indexing and retrieval. We deal with both parts separately in the next two subsections.

7.1 Index Mechanism

The starting point of the indexing process is the collection of documents that needs to be indexed. Before a document in such a collection can be indexed it needs to be parsed. Terrier provides document parsers for several document formats, e.g. .doc, .pdf, .txt etc. The framework is set up in such a way that developers can easily add their own document parser. This is because the framework makes use of a generic document parser, which can be extended according to specific needs. For our project we had to create our own document parser to parse the DICOM-XML documents.

When the correct parser for each document type is defined, the documents in the collection can be parsed. The parser divides a document into blocks of single terms. Before these single terms are passed on to the indexing process, the terms first go through a so-called term pipeline. This pipeline is a modular mechanism that provides the possibility of defining several operations on the terms. These defined operations are then successively applied to each term. The standard operations in the pipeline are stopword-removal and stemming. Operations can be added and removed from the pipeline according to the needs of the application that is developed.

With the terms available from the parser the different index structures can be created. This is done in a two-phase process. The first phase of indexing creates the structures:

- Document Index
- Lexicon (=vocabulary)
- Direct Index

After all the documents in the collection are parsed and indexed and all the above structures are created, the last structure is created in the second phase:

- Inverted Index

These structures are well-known parts of an information retrieval system. We look at how each of these structures is build up in Terrier in a slightly simplified form. The aspect of 'field' information is dealt with afterwards. The different index files are either compressed or fixed-length entry files. Terrier uses advanced compression methods to reduce the size of the indexes. The direct index and inverted index are compressed files, the others are fixed-length entry files.

Document Index This is a fixed-length entry file, where the following information about documents is stored:
- integer document identifier
- document length, i.e. number of indexed tokens for a document
- string document identifier
- ending offset of the document's entry in the direct index

Lexicon This is a fixed-length entry file, where information about the vocabulary of the indexed collection is stored:
- term as a string
- integer term identifier
- document frequency of the term in the collection
- frequency of the term in the collection
- ending offset of the term's entry in the inverted index

Direct Index This is a compressed file, where the terms contained in each document are stored. The direct index is used for automatic query expansion in the Terrier framework. For each document in the collection, the following information is saved:
- term identifier gap (gamma encoding).
- term frequency in document (unary encoding).
- set of flags that indicate whether the corresponding term appears in a specified field

Once the direct index, the document index and the lexicon have been created, the inverted index is created by inverting the direct index. While creating the inverted index also the lexicon is updated with information on the inverted index, i.e. the offsets of the terms in the inverted index. The inverted index has the following structure:

Inverted index This a compressed file, where for each term in the vocabulary postings are stored. For each term in the vocabulary, we save the following information:
- document identifier gap (gamma encoding).
- term frequency in document (unary encoding).
- set of flags that indicate whether the corresponding term appears in a specified field

With the field information it is possible to store information on which fields a term occurred in, in a document. Before indexing starts, the user needs to specify which fields will be recognized during indexing. This can be for example ‘Title’, ‘Section’ and ‘Footnote’. Then during the indexing process to each term a bit string is added. The length of this bit string is the maximum number of fields that will be recognized. When a term occurs in a specific field then the bit at the according position in the bit string is set to 1. In this way it is possible to store for each term a set of fields in which the term occurs. This information can be used in the retrieval process to add restrictions to queries with respect to the fields in which a keyword must occur.

Finally, Terrier is a terabyte retriever. In order to deal with these large amounts of data it makes use of advanced memory management, especially needed for indexing. It provides a mechanism for the user to alter several parameters to control the amount of information that is kept in memory at a specific moment in time, this with a trade-off between the amount of memory used and the total index time. Although also we needed to use these techniques we don’t explain them in further detail, because they didn’t affect the decisions taken in this project.

7.2 Retrieval Mechanism

The starting point for retrieval is the query that the user enters into the application. This query is first parsed according to the query grammar of figure 4. This results in a query syntax tree, that then hierarchically represents the query, where the single terms are on the leafs of this tree.

```
query                   = multitermQuery

multiTermQuery          = (singleTermQuery
                          | fieldQuery
                          | requirementQuery
                          | phraseQuery
                          )+

phraseQuery             = QUOTE_SYMBOL (singleTermQuery)+ QUOTE_SYMBOL

requirementQuery        = (REQUIRED_SYMBOL | NOT_REQUIRED_SYMBOL) multiTermQuery

fieldQuery              = term FIELD_SYMBOL multiTermQuery

singleTermQuery         = term

term                    = (CHARACTERS)+
```

Figure 4: Grammar of the Terrier query syntax

The terms first go through the term pipeline. This is the same pipeline as was used for indexing. This is of course necessary to avoid inconsistent matching. When the terms are out of the pipeline they can be matched against the index. The strategy to transform a query into a ranked set of document is basically as follows:

- Match each term with the index:
  - Lexicon lookup
  - Inverted index lookup
- Assign scores to documents
- Rank the documents

We will describe the matching process and especially the process of assigning the different scores to the documents by using figure 5. Initially we ignore the steps of modified scores, these are the steps where TSMS and DSMS are involved, in order to describe the basic scoring process first.
After the query is completely parsed we process each term of the query. Each term is looked up in the lexicon. When it is found the result of this lookup is the offset to the inverted index where the postings for that term can be found. The postings consist of a list of documents in which that specific term occurs. So reading the postings leads to a set of documents for a particular term. Each document is then assigned a score for that term. The calculation of the score is done based on the weighting model that is used. Each weighting model makes use of the collection statistics as shown in figure 5 that

consist of term frequencies and document frequencies. More on these weighting models will follow later.
This process is repeated for all terms in the query and after that for a certain document *x* the first step of assigning scores of figure 5 is finished. The total score of a document is then the sum of the scores for the different terms.
After all terms of a query have been processed, a number of the documents from the collection has been assigned a score. Based on these scores we can rank these documents so that we get our ranked result set. Now we have described the basic concept of assigning calculating a document score.

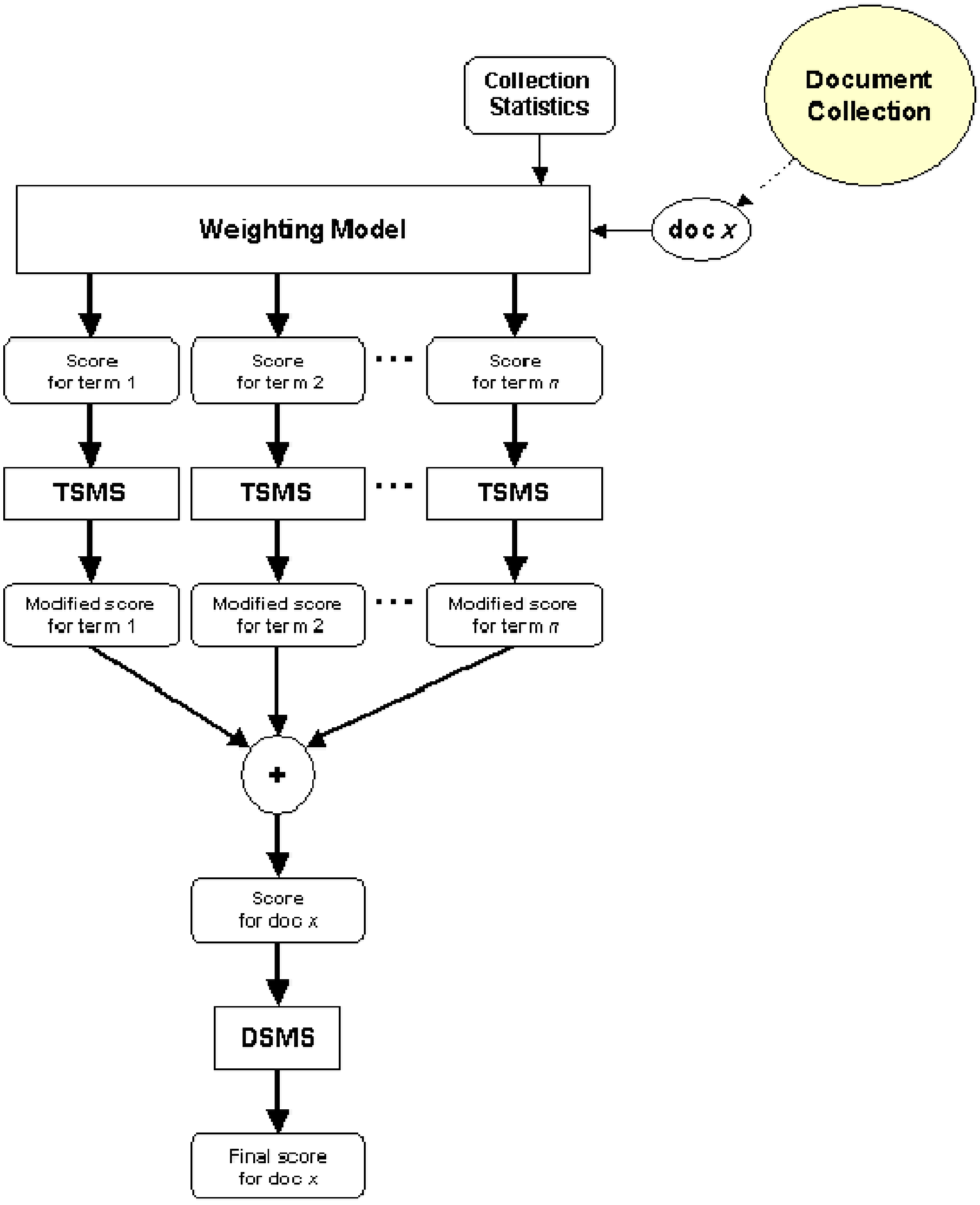


Figure 5: Process of assigning document scores

Term and Document score modifiers

We continue by describing the *term score modifiers* and the *document score modifiers* that are represented by TSMS and DSMS respectively in figure 5. These two mechanisms play an important role in assigning scores to documents and therefore affect the way that the documents are ranked.
The *term score modifier* makes it possible to alter the score of a document for a specific term after the score has been assigned by the weighting model. An example of how this modifier is used, is when

the query contains a term with the restriction that this term must occur within a specific field. After the documents have been assigned a score for that term, the *term score modifier* modifies (i.e. resets) the scores for documents where the term does not occur in that specific field.
The *document score modifier* operates on the complete result set of documents after all the terms of a query have been processed. A *document score modifier* modifies the total scores of documents and the effect of a *document score modifier* is that it decreases the size of the result set of documents and/or changes the order of ranking. An example of a *document score modifier* that is used by default in the Terrier framework is the 'boolean fallback modifier'. The principle of this modifier is that when there is at least one document in the result set that contains all the terms specified in the query, then the score of all documents that contain only some of the terms is reset. In other words: the standard boolean constraint for a query is 'AND', when this is not feasible then the constraint is relaxed to 'OR'.
For both the *term score modifier* as the *document score modifier* holds that it is easy to implement your own modifier and then use it in the framework. In our project we made use of modifiers that were provided by the Terrier framework, like for example the 'boolean fallback modifier'. Also did we implement are own *term score modifiers* and the *document score modifiers* to support our different types of queries that we specified. It will be discussed in the implementation part.

Weighting models
A concept that is characteristic for Terrier is that it uses the Divergence From Randomness (DFR) framework. Several weighting models are implemented in the Terrier framework. Two of them are classic IR models, the others are parameter-free probabilistic models according to the DFR framework. The DFR approach is based on the simple idea: "The more the divergence of the within-document term frequency from its frequency within the collection, the more the information carried by the word *t* in the document *d*". This means that the term weighting models are obtained by comparing the actual term distribution with the distribution of a random process. More about these models can be found in [20].
All the implemented models have their own characteristics and for our project the experiments needed to determine which model performs best, if there are differences at all.

7.3 Conclusion

The Terrier framework is flexible starting point for our DICOM retrieval system. Its modular design allows us to easily modify the framework. These modifications were mainly needed to make the system able to support our specified types of queries. It has led to changes in the index structure that are used as well in the retrieval and weighting process.

8 Related Work

We have chosen to use the Terrier platform as our base system to deal with the retrieval of the DICOM XML documents. A lot of research in the field of XML retrieval in general has been done, which was also mentioned as one of the advantages of using XML. We can divide the approaches in this field roughly into three classes: RDMS systems, native XML databases (NXD) and information retrieval systems. The Terrier platform obviously is placed in the third category.

RDBMS
Several attempts have been done to use RDBMSs as (the basis for) an XML retrieval system. The big advantage for our problem would be that these systems are extremely suitable to support data retrieval queries. There are however some difficulties when storing XML documents in a RDBMS. XML documents can be stored in RDBMSs in two ways, either as documents as a whole or decomposed [21]. Storing documents as whole is not an option, because during retrieval these documents have to be sequentially searched through which will lead to performance problems. So the XML documents have to be mapped on the relational model. In most of the cases it is possible to map the XML structure on the relational model [22]. However, to correctly make this mapping is laborious in our case because there is no schema of the XML documents available and the document structure can change. Further when the XML data has a regular structure, but that structure varies a lot like in our XML data, mapping it to a relational database results in either a large number of columns with null values (which wastes space) or a large number of tables (which is inefficient) [23].

Native XML databases

The definition of a native XML database is according to [24] a system:

- that is specialized in storing XML data and stores all components of the XML model intact.
- where documents go in and documents come out
- that contains a full implementation of XPath or XQuery

The advantage of these systems is that they are designed for querying structure in combination with content. Further, some of these systems like exist [25] support storage of schema-less XML documents. However, these systems usually don't provide data retrieval queries and ranking mechanisms because these two aspects are not part of the specification of XPath or XQuery.
There are exceptions, for example XIRQL [26], which is an extension to XPath based on information retrieval concepts and is implemented in HyReX [27]. It supports ranking of the results and the use of predicates like '<' and '>' on the data types that are used. However, this language requires XML documents to conform to a predefined DTD. This is also described as a drawback in [28].
Finally, we don't require the full capabilities that XPath or XQuery offer. One could imagine that using such an implementation would introduce unnecessary extra (index) overhead.

IR systems
Besides Terrier there are not many open source IR systems that are freely available. Further most of them, including Terrier, are designed for natural language and simply ignore numeric data. Therefore data retrieval queries are not possible. Besides Terrier, Lucene [29] is a mature IR system that can make use of structural information in querying [1].

Besides the systems in these different categories, there are systems that overlap the categories. In this context [30] states that, considering the wide variety of semi-structured data available in structured XML documents, a need exists for a system that can provide database-like precision when searching typed content within an XML document. A system that supports this type of searching should also still be able to provide the functionality for the user to search the full content of documents and provide a list of matching documents to the query ranked by relevance. Now, regardless whether this is true in general, it is indeed true for our project.
Therefore in [30] a hybrid approach for structured document retrieval is described. The motivation for this approach is to integrate:

- approximate matching
- exact matching
- range matching

In [30] is further described that current projects that focus on IR from structured document are still in their infancy and do not consider the problem of creating a system that supports exact and range matching of document content.
The hybrid approach in this report means that it makes use of both an RDBMS as an IR system. In the report a plan for such a hybrid system is proposed, however no implementation has been done so no test results of this plan can be described. Interesting would be what the impact is of the overhead that the combination of the two different systems brings about. This compared to solutions that use only one type of system.

In [31] one of the few attempts is made to support range queries in combination with IR from XML documents. In order to accomplish this, extensions are made to the standard index structure of the information retrieval system to support queries that specify numeric ranges. The given reason for this extension is that in information retrieval systems normally treat numbers as keywords; a user searching for "1999" can find the exact value "1999", but it is not possible to search for "between 1000 and 2200".
To enable numeric queries, when the implementation of this application encounters a number it stores it as several *pseudo-keywords* representing different numeric ranges which contain the target number. For example, "1999" might be indexed under the pseudo-keywords "1000-2000", "1900-2000", and "1990-2000", as well as the actual value "1999". The problem with this solution is of course that you need to determine the correct pseudo-keywords in advance. When choosing the ranges too large, users are returned too much irrelevant results (precision decreases). When choosing the range too

[1] The latest version of Lucene, which was released in February 2006, now does support data and range searches. At the moment of the start of our project there was not yet such a system available.

small, users are returned less result (recall decreases). To choose the right set of pseudo keywords is for our problem impossible.

9 Global approach description

Based on the previous sections, in this section we will introduce our approach by giving a global overview of the total system. Also we introduce the most important concepts that are used by the system. The next sections then will go into the approach in more detail.

The decisions that we discussed up till now are that we use a combination of DR and IR, that we use XML documents to represent the DICOM images and that we use Terrier as our base IR system. Based on these decisions and on the other requirements of section 3 we have made a design for our system.
figure 6 schematically shows the architecture of our system. The central point of our system is a modified and extended version of the Terrier retrieval platform. This part will deal with the actual indexing and retrieval of the DICOM XML documents.

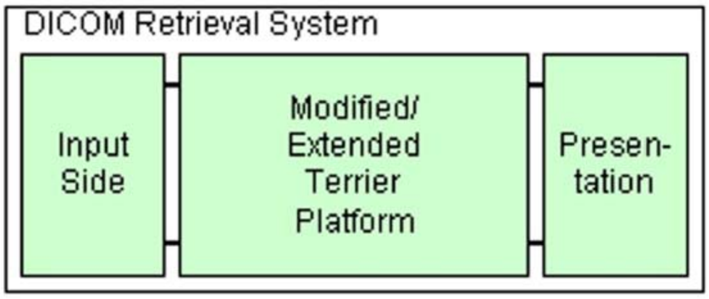


Figure 6: Schematic overview of the DICOM retrieval system

Besides this modified version of Terrier we have added a part at the input side of the system and a part that will deal with the presentation of the information to the user. The input part is responsible for the storage of the DICOM images and the automatic generation of the XML documents. Further contains this part our parser that deals with the parsing of the DICOM XML documents.
The presentation part of the system provides a user interface to the system. It deals with the way the queries can be entered into the system and the way that the returned results are shown to the user. The results that are returned by the retrieval engine are a set of DICOM XML documents, because these are the documents that are indexed by the modified Terrier platform. The user however is interested in retrieving images. The presentation part makes sure that the returned XML documents are linked to the corresponding images. Further contains this part the functionality of showing previews of the DICOM images.
The next two sections deal with the details of the implementation of the system. First the modifications and extensions to the Terrier platform are discussed in section 10 and in the next section the other system implementations are described.
We continue this section with the discussion of two important concepts of our system that are used in the description of the implementation. These concepts are the distinction between comparable and non-comparable term and the use of semantic tag names.

Comparable vs. Non-comparable
Simply put, a comparable term is a term that occurs as the only term within a tag. Non-comparable terms are the opposite. These are terms that occur in tags that contain multiple terms as in sentences or text.
It is the responsibility of the parser to mark terms comparable or non-comparable and this information is maintained in the index.
The reason to make this distinction is twofold. First, the decision is related to the relevance of the results to a data retrieval query. A lot of the DICOM data is of the form *tag-value*. The data retrieval part of a query is of the form *tag-operator-value*. These query parts can look for example like `0<errors<10` and `version==1.4`. The single term within a tag then is the value that needs to

match the value part of the query. We believe that the same queries applied to tags that consist of multiple terms (=sentences) would have practically no informative value with respect to the operators we use.
The other reason is that this distinction can lead to more efficient retrieval from the index. We are dealing with an enormous amount of data; data that can be queried with data retrieval queries. We want to keep the subset of data to search through as small as possible. By making already the distinction between comparable and non-comparable data at indexing time, we keep this set as small as possible.

Semantic Tag Names
In the requirements we described that the structure of a DICOM image, and therefore a DICOM XML document, can be used for querying. Because we use a tool to generate this XML, it is machine generated data, although it is mostly human readable.
As a consequence of being machine-generated, a lot of the XML element names don't have an intuitive meaning. However the requirements described that no need for extensive knowledge of the underlying structure was required for the user. So we cannot by default use the element names as they occur in XML files for indexing. To illustrate this in figure 7 we return to a simplified fragment of a DICOM XML document:

```
<Attribute Group="0018" Element="1151" Name="X-Ray Tube Current">
       603
</Attribute>
       . . .
<Attribute Group="0028" Element="0004" Name="Photometric Interpretation">
       MONOCHROME2
</Attribute>
       . . .
<Attribute Group="0008" Element="2111" Name="Derivation Description">
       Compress BN JPEG Lossless, Decompress Pegasus JPEG Lossless
</Attribute>
```

Figure 7: DICOM XML fragment

Here we see another problem. Not only are the XML elements semantic-meaningless, also can a lot of different parts all share the same element, which makes it useless for using in queries.
The problem of semantic meaningless tag names is not unique for our domain, but it occurs in many other fields where XML is used to represent the data. For example in [32] the problem is described as well as the need for properly indexing the tags in order to avoid false negatives during retrieval.

What we want is that in the example above, when we use structure in our queries, that we are able to use names like 'Photometric Interpretation' and '00280004' to indicate the structural parts we want to query.
It is the responsibility of the parser that these semantic tag names are created correctly and in the index they must be processed and maintained. The corresponding sections on the parser and index deal with these mechanisms respectively.

PART III

10 Terrier Modifications and Extensions

This section describes the modifications and extensions that were made to the original Terrier framework. In analogy to section 7 where the Terrier framework was discussed we have divided this section into a part on indexing and a part on retrieval. Besides these two parts we also have a part where the necessary modifications to improve the performance of our system are discussed.

10.1 Index

In this section we describe the index process of our DICOM retrieval system. The structure of this section is the following. We first deal with the requirements that our index structure needed to satisfy. Then we compare these requirements with the index structure of the original Terrier framework and we define what we had to add and modify. Next we deal with the details of the implementation of our index solution and finally look at the effects of our approach.

Index requirements
The requirements for the index structure are based on the requirements that are described in section 3. Also these index requirements can we divide into two parts. We have requirements for the lookup functionality that the index must provide. This is the lookup functionality that is needed to support the several types of queries have defined. On the other hand we have requirements on the way the index is created.

The requirements that our index had to satisfy are the following:

Lookup functionality

- Lookup of a term.
  Given a term it must be possible to enumerate all documents in which the term occurs.

- Lookup of a term in a specific structural context.
  Given a term in combination with a tag it must be possible to enumerate all documents in which the term occurs within the specified tag

- Lookup of a term in combination with a predicate and a tag
  Given a term in combination with a predicate and a tag it must be possible to enumerate all documents where terms satisfy this condition and occur in the specified tag

Index creation functionality

- Index tag context along with the term
  The relation between terms and tags must be maintained

- Index unseen tags on the fly
  New unseen structures must be indexed dynamically

So we have three lookup requirements to support our different types of queries. We need to be able to query on terms only and on terms that should occur in a specific context, i.e. the containment queries. Finally we have the data retrieval queries. These queries contain a condition that consists of a keyword, a tag and a predicate. These predicates can be ‘<’, ‘>’ or ‘=’.
For the creation of the index we have requirements on the way that the structure of the documents is indexed. We have defined queries that use the relation between terms and tags. It is therefore necessary that the tags are stored along with the corresponding terms. Further we discussed that we need flexibility with respect to the document structure. For this reason it is necessary that it is possible to index new unseen tags on the fly. This is needed because before the indexing starts it is not known which tags will be encountered in the different documents.

Next we compare our requirements with what is already available in the original Terrier framework. When we look at this framework we see that already available is the lookup of terms and the lookup of terms in a specific context. Further is it possible to store structural information along with the terms. So the two aspects that need to be added are the lookup support for the data retrieval queries and the functionality to index new unseen tags on the fly. Besides this, the index needs to deal with the difference between comparable terms and non-comparable terms that we discussed in the previous section. These two are related to each other; to support the data retrieval functionality the index must be extended with the lookup for exact matching and range matching and the concept of comparable terms.

The next parts describe how this functionality is added. First we discuss the organization of the index. We look at the different structures that the index consists of and what they look like. Then we look at the aspects of tag indexing and finally we look into details of the ordering of the lexicon. This ordering plays an important role in the lookup of range queries.

10.1.1 Index Structure

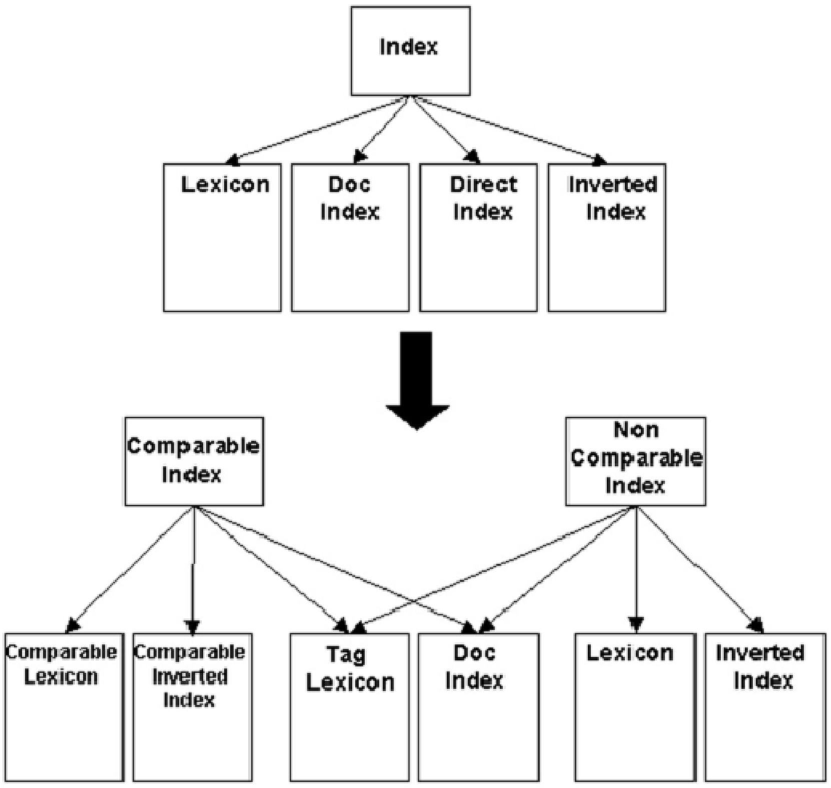


Figure 8: Index change

Here we will discuss the organization of the index that we have used. For this we use figure 8 as our guide. This figure represents the change in the index structure from the original Terrier framework into the structure that we have used. The different parts of the original index are discussed in section 7.1.

Index split up
The first aspect that has changed in the new index structure is that the index is split up into two parts: a comparable index and a non-comparable index. This distinction was made to maintain the difference between comparable and non-comparable terms like we discussed in section 9.
Splitting up the index was one out of two options we regarded to maintain the distinction between comparable and non-comparable terms in the index. The other option was to set a flag for each posting in the inverted index that indicates whether it is comparable or not. Although this option seems simpler than our solution, it has a drawback. With that solution, in case of a conditional query, the terms are retrieved from the index and afterwards the non-comparable terms are filtered out based on this flag. In section 9 we explained that one of the reasons to create the distinction between

comparable and non-comparable terms was to make sure that a smaller subset of data is processed. That is the reason that we decided to split up the index. Instead of post-filtering the results based on a flag, with our solution in case of a conditional query we directly work with a set of only comparable terms.

Next we explain how our index is split up. Splitting up the index means splitting up almost all structures of the original index. The document index does not need to be split up, because it only contains information at document level, and not at term level. The document index can be shared by the comparable index and the non-comparable index.
So initially we get two direct indexes, two lexicons and two inverted indexes. In figure 8 however we see that in the new situation we don't have a direct index. This is because in the original framework, after the inverted index is created, the direct index is only used for automatic query expansion. Query expansion can be used to improve the retrieval results. This was not necessary in our case so we did not use the automatic query expansion functionality. Therefore in our application the direct index is deleted after the inverted index has been created.

The lookup from this split index structure is as follows. When a term needs to be looked up it is first determined whether we are dealing with a comparable term or not. This depends on the type of query. In case of a data retrieval query, the term is a comparable term and needs it to be looked up from the comparable index.
Of course this introduces a penalty. In case of a 'normal' query, i.e. a content only query, both indexes need to be consulted to get the set of result documents. This is because both indexes can contain postings for a particular term. When both indexes contain postings for that term then the statistics for that term – document frequency, term frequency in a document and term frequency in the collection – that are used for weighting need to be merged, i.e. need to be added.
Also will the index grow in size. This is due to the fact that a term can occur in the collection both as comparable term as non-comparable term. The worst-case scenario is that all terms occur as well as comparable term as non-comparable term. Then the total lexicon grows twice as big. The experiments have shown that the total index size for our application is 20% of the total collection size. This is an acceptable size for our project.

In section 7.1 on the Terrier framework we described the aspects of the indexing process of the Terrier framework. Documents are parsed and then the terms are indexed. Before the terms are indexed they first go through a pipeline where, amongst possible other things, stemming and stopword removal are applied.
Now when we are dealing with exact matching, we argued that this is only applicable for comparable terms. However when stemming is applied to these terms while indexing, then it is possible that these terms are changed. For exact matching this term will result in a mismatch, which causes a query to not produce results although there are documents satisfying the condition. Therefore we don't apply stemming for comparable terms. Also stopword removal is not applied for the same reasons.
Now we have made sure that the comparable terms are suitable for exact matching. However this data can also be queried using content only queries, because in that case both indexes are consulted. In a content only query, stemming is applied. This would therefore lead to a false mismatch when terms are looked up from the comparable index. To solve this, when a term is comparable and would need stemming according to the stemming rules, we store the unstemmed term in the comparable index and the stemmed term in the non-comparable index. This way, by maintaining also the stemmed version of the term, although it is a comparable term, we have not reduced the information retrieval capabilities of the application for content only queries. Stemming supports the fuzzy matching of an information retrieval system. This approach is an important aspect for the integration of data retrieval and information retrieval, because it enables both types of matching.
Of course this approach again causes the total index to grow, but this increase is relatively small because the comparable terms tend to be terms where stemming and stopword removal often don't need to be applied.

We already discussed the removal of the direct index. Further we see in figure 8 that we have an inverted index and a lexicon for both the indexes and that we have a shared tag lexicon. This tag lexicon is a new introduced structure. It is used to facilitate the indexing of structural information. It will be discussed in the next subsection that deals with the tag indexing. Also the changes that were needed in the inverted index to correctly support the indexing of tag information will be discussed here.

The changes that were made in the lexicon are then described in the subsection on the lexicon ordering.

10.1.2 Tag Indexing

This subsection describes how the structural information of the documents is indexed. The requirement is to support on-the-fly tag indexing to support schemaless XML documents. The tags need to be indexed and their relation with their content, the terms, must be maintained to support queries that combine content and structure.
To meet the requirements for indexing structural information we first looked at the way the Terrier framework deals with what is there called the field information. This is described in section 7 on the Terrier framework. With this approach there are two problems for us. First the bitstring that Terrier uses to store the tag information is of fixed-length, which is determined by the number of tags that are defined in advance by the user. So new tags cannot be automatically processed.
Second, because of this approach, the length of the bit string is equal to the total number of fields available in the Terrier framework. It is designed to work only with a small number of fields; this in contrast with our problem, where we are dealing with thousands of different tags. A term always occurs in the context of a few of these tags. A fixed length bit string with a large length would therefore be used very inefficiently with respect to the space it takes up in the index.
The advantage of the approach of the Terrier framework is that the tag information is directly available with the term. The bit string directly corresponds with a list of predefined fields.
We decided use a similar approach. We have changed the structure of the direct and inverted index so that for each term in the index instead of a bit string a sequence of tag identifiers is stored. This sequence of tag identifiers is variable. Every time a new tag is encountered, the tag is assigned an identifier and the tag together with its identifier are stored in the tag lexicon. This tag lexicon consists of pairs of tag names and identifiers and it is ordered according to the tag names.
The identifiers of the tags in which a term occurs are stored with this term in a sequence. This is an ordered sequence of tags where the first identifier is the identifier of the tag that directly contains the term. This way the hierarchy of terms is preserved. The sequence of tag identifiers is encoded and compressed as a sequence of tag identifier gaps.

10.1.3 Lexical Ordering

This section describes the changes that we made to the lexicon. The reason for us to change the lexicon was to support the lookup of a range of terms. This is necessary for our range queries.

To lookup a range of terms we use the lexicon to lookup which terms fall into this range. This is possible because the lexicon is always an ordered list of terms. To make this lookup function correctly, the ordering needs to meet specific requirements.
The basic Terrier lexicon is, like a vocabulary is supposed to be, lexicographically ordered. In a lot of IR-systems, as well as in Terrier framework, numbers are ignored and often also number-like terms. However for our retrieval application we also need numbers and number-like terms for retrieval. An example of a number-like term is the Unique Identifier (UID) of a DICOM image that could look like ‘12.34.83.346.54412’.
To lookup a range from the lexicon, first the positions in the lexicon of the upper boundary and the lower boundary are determined and then all terms in between belong to that range. The problem with the lexicographic ordering is that numbers are not ordered according to their numerical value. For example 12 will precede 5, where it should be otherwise to have an intuitive range lookup. Further for the example of UID’s, we would like to have the situation where “1.2.44.3.11.12.60.344” precedes “1.2.44.3.11.4.75.301” where this is lexicographically not the case.

In order to support this functionality we have changed the ordering of the lexicon from a pure lexicographic ordering to an ordering in which numerical values are taken into account. The modification means that the numbers are ordered based on their numerical value. Further, terms that contain a numerical part are ordered both lexicographic as numeric, i.e. terms are first ordered lexicographically but when the order of two terms is based on the numerical part then this is based on the numerical value. For example a term like “Siemens4CT” precedes “Siemens1000CT”.
Now that we have our desired ordering, we have to make sure that this ordering satisfies the requirements of a lexicon. First for building the lexicon and for lookup from the lexicon the same

lexicographic ordering function must be used. In the original lexicon this functions correctly because the ordering function is reflective, transitive and symmetric. Without a formal proof we can state that for the modified lexicographic ordering this is also the case.
The type of ordering that we used is also used in other applications, e.g. the newer versions of Windows Explorer also use this ordering to order filenames.

The effect of this type of lookup is that for a range lookup we implicitly regard all terms to be of the same data type. This as opposed to classic relational data retrieval systems where we have values with different fixed data types. When for a specific tag a range of values is requested then the corresponding column in the correct table is consulted and the range of terms is retrieved. All of the values in a column are of the same type. In a system that uses typed values, each type can have its own specific ordering. This is according to the definition of a data type in information retrieval systems of [33]:

> A data type |D| is a pair (|D|, P_D), where |D| is the domain and P_D ={p1, . . ., pn } is the set of (vague) predicates, where each predicate is a function pi: |D| x |D| → [0, 1].

In our case we have one generic data type where we use the predicates '<' and '>' on, based on the ordering described above. This ordering is relevant for the major part of the field-value pairs in our collection. However there are of course some type of values that require their own specific ordering and therefore cannot be correctly retrieved using our approach. An example of these values are the 'ages'. In the DICOM standard age is represented as `xxx{D|M|Y}`, where D stands for days, M for moths and Y for years. The value '12D' is ordered after '2Y', which is obviously not correct when interpreting the values. This a kind of functionality that is lost and it is a side-effect of the trade-off between flexibility and functionality.

10.2 Retrieval

In this section we deal with the aspects of the retrieval process of our application. We do this by discussing the details of the different retrieval mechanism that are needed to support the different types of queries that we have defined.
The aspects that we discuss in this section are keyword retrieval, containment retrieval and data retrieval. Further we deal with the concept of taking the meta files into account during the retrieval process. Finally we discuss the weighting details during the different retrieval mechanisms.

Keyword Retrieval
We start with the retrieval of a keyword query. The basic concept for this type of lookup has not changed with respect to the original Terrier framework. The only aspect that has changed is that now this lookup has to be done from the split index structure. The process for the retrieval of documents for a specific keyword is as follows. The keyword first goes through the pipeline that has been used in the indexing process. Then the keyword is looked up in both the comparable lexicon as the non-comparable lexicon. The entries, if they have been found are used to read the postings from both the inverted indexes. This leads to two sets of resulting documents. These two sets of documents are merged into one set. Documents that occur in only one of the set are directly placed in the new set. Of the documents that occur in both sets first the statistics are added and then the documents are placed in the new set. Once this set is created the normal assigning of scores to the documents can be applied.

A mechanism that in a slightly different form was also available in the original Terrier framework is the modification of a score for a keyword depending on the context that it appears in. For this purpose we make use of a *term score modifier* that was already discussed in section 7.2. In the configuration of the application different tags can be defined for which the score of a keyword will be altered. Also in this configuration the multiplier must be defined with which the original score is multiplied when the keyword occurs in such a tag. The 'context modifier' is a *term score modifier* that checks whether a keyword that has a document as a result occurs in one of the specified tags. If this is the case it multiplies the original score with the defined multiplier. This mechanism allows the administrator of the system to assign a relatively higher relevance to specific structural parts of the DICOM XML documents. The tags to be defined have to determined based on the expertise of the users and the value of the multiplier has to be empirically determined.

Containment Retrieval
For the retrieval of containment queries the retrieval process is a follows. First the term is looked up in the way that is used for the lookup of keywords. Like we described this leads to one set of documents that are assigned a score for that term. After that a *term score modifier* is used that resets all scores to zero for the documents in which the term did not occur in the right tag. This mechanism is the equal to that of the original Terrier framework, except that instead of a bit string now a sequence of tag identifiers must be read to determine whether a term occurred in the right tag or not.

Data retrieval
The retrieval process for exact match queries is almost equal to the scenario described for the retrieval for containment queries. The difference is that the term that is used in the query doesn't go through the pipeline like normal keywords. The unchanged term is used for the lookup. Another difference is that this term is looked up only in the comparable lexicon. The rest of the process is equal to that of the retrieval of containment queries.
The process for the lookup of range queries is related to the lookup of exact match queries. The first step in the process is that the upper and lower boundary of the range are looked up in the comparable lexicon. After this step the postings are read for all terms in the comparable lexicon that lie between these boundaries. Also for this resulting set of documents the documents are filtered out in which the terms did not occur in the requested tag.

Meta documents linking
In the requirements we stated that we are not interested in retrieving the documents containing the meta description of a dataset of single image itself, but that the information contained in this documents must be taken into account during retrieval. To achieve this we increase the score for documents of which the meta XML document contains relevant information. When a meta XML document matches a specific query then this document will appear in the resulting set of documents. We use a *document score modifier* to process the meta XML documents in this result set of documents. This modifier filters out all meta files from the result set. Then each of the meta files is linked to the documents that they belong to. This is possible due to the way the documents are ordered in the file structure that will be discussed in the next section. Then after the meta documents are linked the score of the documents that the meta document belongs to is increased. These can be documents that were already in the result set and therefore already have a score or it are documents that still had a document score of zero. The scores of the documents that belong to a specific meta document are increased by the score that the meta document has been assigned multiplied by a constant. This constant has to be empirically determined.

Weighting aspects
Several types of weighting mechanism are proposed for ranking structured documents. The first is the basic tf*idf, known from the classis IR which is used in [34]. In this case the documents are treated as atomic units. Some researchers [26, 35] argue that in the context of structured documents not the documents itself should be treated as atomic units. Instead, the separate elements/parts that are used for indexing should be used as atomic elements and then at this level a kind of tf*idf should be applied. However we are interested in complete document retrieval, not parts of the documents. This is the reasons that also in our project we have treated the documents as atomic units. In the this way we were able to the weighting mechanism of the original Terrier framework.

For the data retrieval queries the documents are not assigned a score. Instead, when a part of a query consist of a data retrieval query then this part is the filter for documents to appear in the final set of resulting documents. Only documents that match the conditions of the data retrieval query parts will appear in the result. In the case that a query only consists of data retrieval parts then all documents are returned that match all conditions.

We believe that a document is most relevant when in case of a containment query, the required term is contained in (a more) direct containment. We illustrate this with an example:

Query: $\texttt{tag}_x$ must contain `monochrome`

Document$_1$:
```
<tagₓ>
 <tagₘ>
  <tagₙ>
```

```
          monochrome
      </tag_n>
     </tag_m>
    </tag_x>
```

Document$_2$:

```
    <tag_x>
          monochrome
    </tag_x>
```

In this example we state that document$_2$ is more relevant with respect to the given query because the term occurs in a more direct containment. Actually what we want is that the score of a term is down-graded when it is propagated upwards in the hierarchy. This mechanism is also described in [26]. Here a so called ‘augmentation weight’ is used to multiply the score with each time the score is moved upwards in the hierarchy. The value of this multiplier has been empirically determined to be 0.7. To illustrate this with an example, we look at above given fragments. Further the score for ‘monochrome’ in document$_1$ is 0.9 and for ‘monochrome’ in document$_2$ is 0.8.
The score for the second document will remain 0.8, but the score for the first document has to be downgraded twice, resulting in 0.9*0.7*0.7=0.441. In this case document$_2$ will indeed be ranked before document$_1$.

10.3 Performance Optimization

This section describes some modifications we made to the index structure in addition to the previous sections in order to improve the retrieval speed.

The reason that we looked for performance optimizations is that the retrieval process can be very inefficient, especially for range queries. This is caused by the way this process is set up. Like we described in the previous section, when a combination of a tag and a value is queried the value is looked up in the lexicon and then the postings for that lexicon entry are read from the inverted index. From the set of resulting postings all the postings in which the value did not occur in the correct tag are filtered out. This means that these postings have been looked up unnecessary. Some ‘popular’ values like 0 and 1 occur in lot of different tags. So querying these values leads to a lot of miss hits. This problem increases when for a range of values is queried and then it also leads to performance problems of the retrieval application.

To solve this problem we have created an extra index structure that consists of postings for combinations of terms and tags. Actually it is an inverted index for these combination of terms and tags. From this so called inverted term-tag index postings for combinations of terms and tags can directly be read. A combination of a term and tag is only placed in this index when the total frequency of the term is above a specific threshold. When the total frequency of a term is relatively high then it will probably occur in a lot of different tags and is it beneficial to store the combination of tags and these term in this inverted index. The value for this threshold has been determined by the experiments. The reason that we use a threshold is that for terms with a relative low frequency the performance improvement is small and would therefore cause the index unnecessary to grow.

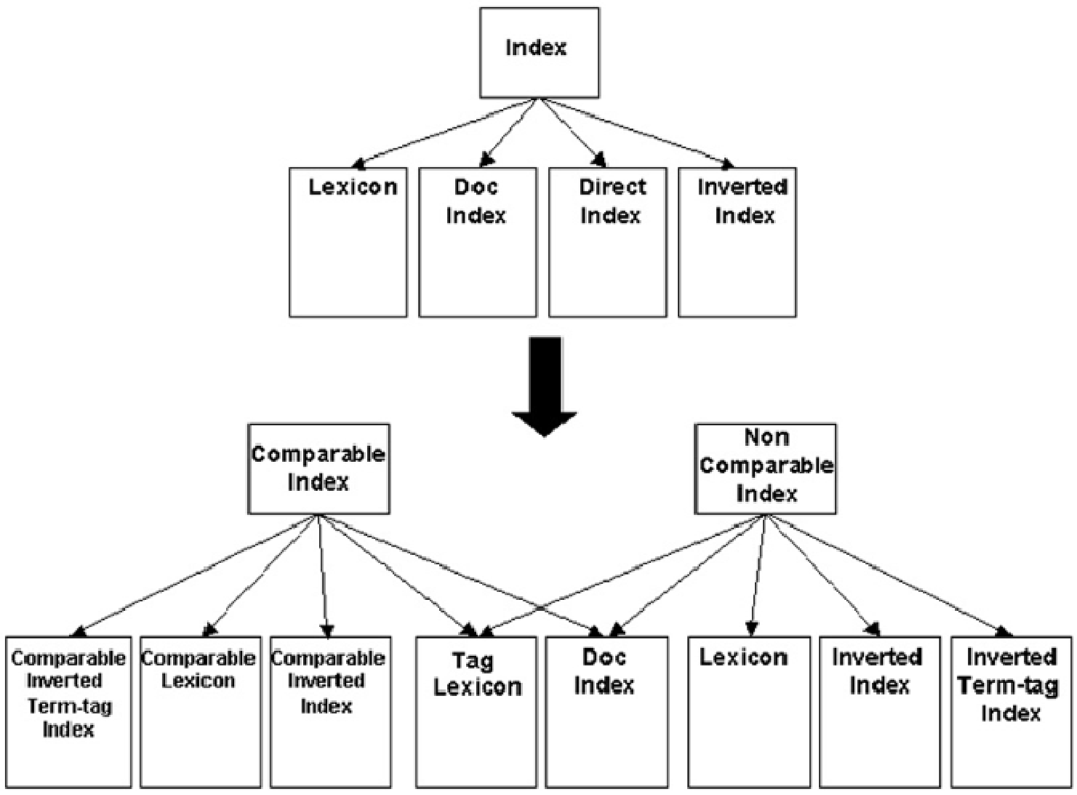


Figure 9: New index structure

When now for a term is queried in combination with a tag then first this term is looked up from the lexicon. In the lexicon then the total frequency for this term can be read. When it is below the given threshold then the lookup is done based on the 'normal' process. When the total frequency is above the given threshold then the postings for the combination of the term and the tag are directly read from the inverted term-tag index. In figure 9 the new total index structure is shown. We see that also here the division between comparable and non-comparable terms is maintained. That means that the non-comparable inverted term-tag index will only be used for containment queries.
The inverted term-tag index is created after the inverted index is created. This means that a third phase in the indexing process is introduced, after the phases of the creation of the direct index and the inverted index. In this third phase all the terms with a frequency above the given threshold are selected and their postings are read from the inverted index. Then all possible combinations of those terms with a tag are extracted and these are placed in the inverted term-tag index.

11 System Implementation

This section describes the implementation aspects of the DICOM retrieval application that can be regarded to be outside the domain of the original Terrier framework. This section will cover the input side and the presentation side that were presented in figure 6 of section 9. For this purpose, in this section we first deal with the aspects of the parsing of the DICOM XML documents, which we regard to be at the input side of the application. Then we describe the aspects of the user interface of the application. This is divided into two parts; a part on the user interface that provides the functionality to load DICOM objects into the system and a part on the user interface that provides the functionality of query input and presentation of the results.

11.1 Document Parsing

In section 9 where the application was globally described already two responsibilities of the parser were mentioned. These were to make the distinction between comparable and non-comparable terms and to create semantic tag names. How the parser provides this functionality will be discussed in detail here.

The parser that we used to parse our DICOM XML documents is a StAX (Streaming API for XML) parser. StAX is the latest API from Sun's JAXP family [36]. StAX was created to address the limitations of DOM and SAX.
For our problem we need a fast simple parser. The XML documents to parse are automatically generated by a tool (DVT) so we can assume that they are well formed. Also we don't need to validate them against a schema. SAX would therefore be, compared to DOM, the best solution.
A SAX parser is an event driven parser. While parsing it generates events of items that it encounters. For the framework we are using however a 'pull parser' would be more suitable. StAX parsers are also pull parsers so therefore we use these. We use the Sun implementation of the StAX parser.

The basic functionality that the parser must provide is to divide the documents in separate terms. These terms are one by one delivered by the parser to the indexing mechanism. Like we stated these terms have to be marked comparable or non-comparable based on the given conditions.
Not only must the parser provide the different the terms of a document, it must also provide the structural information for each term. To provide these structural information, during parsing the parser maintains a stack of all the tags it has encountered. This way for each term the information is available in which tags the current term is contained. By maintaining a stack of tags the order of the tags is preserved.

Semantic tag names
For these tags in which the terms are contained we want to create semantic names. Current solutions to create semantic tag names in XML documents vary from a set of rules according to which the names are created, to complete self learning systems. Because the diversity in the structure of the documents in our collection is not very high, i.e. a lot of structural parts have the same pattern, a set of rules to define the parse behavior is sufficient for our project.
In [32] a fully configurable XML index approach is taken. First all characteristics of the data in the collection are collected by the system into an Excel sheet and then this sheet is presented to the user. Based on this Excel sheet, the user defines an index configuration and based on this configuration the document collection is parsed and indexed. This approach would be too laborious for our project, so we use a derived approach. We have used a configuration where users, most probably the system administrator, can use three types of parse rules.

- tag names that will not be maintained in the index, but of which the content is indexed
- tag names of which the tag and everything that is contained in the tag is ignored
- rules that prescribe the construction of (semantic) names to be used in the index for defined elements

The default values of these rules are defined by experiments but they can be adjusted and extended easily.
The first two types of parse rules are meant to provide a more efficient use of the index. These parse rules make it possible to remove unnecessary parts from the index. An example of this kind of information is a lookup table that is sometimes contained in the DICOM XML document. It can contain information that is used by an image application. The data in this lookup tables then consist of ranges of values that these applications use for correctly showing the image on the screen, but it will certainly never be queried.
The third type of parse rule describes how the semantic tag names are created. For this we return to figure 7 of section 9. Here we gave an example of how a semantic tag name could look like. In general we want to replace the name of an XML element by a name that has a semantic meaning and therefore can be used by users in their queries. This is however not for every element the case, because some elements already have a semantic name. We found that the semantic names often can be found in the attributes of an XML element, in attributes like "name" or "id". We use the attributes of an XML element to create semantic tag names. In the example of section 9 we notice two things. First is stated that we want to replace the existing element by two names instead of one, namely 'Photometric Interpretation' and '00280004'. More generally, element names can be replaced by multiple new names. Secondly, the new tag name '00280004' is a combination of the values of the attributes 'Group' and 'Element'. Attributes can be concatenated to form one new tag name.

The rules for creating the semantic tag names to use in the index consist of a set of mappings. Each mapping consists of a relation between an XML element name and a set of attribute names of that

element that must be used for that element to create the tag names to index. In this relation is also indicated which attributes should be concatenated to form one tag name. Further can attributes be specified that will be used in general. This means that regardless what name an element has, always is looked for those attributes to construct a new tag name.
In section 10.1.2 we described the mechanism to index the tag names. Each new tag is assigned a unique identifier when it is encountered. Now in case that an element name is replaced by multiple new tag names, then all these tags are assigned the same tag identifier. In this way these names are all seen as the same tag during the retrieval process. So regardless which tag name the user uses in his query, it will all lead to the same result, because with this identifier internally still is dealt with only one tag.

The effect of this approach is that different DICOM XML structures can be used in one collection. We said that when an element is renamed to multiple tag names that they are assigned all the same identifier. The process for assigning an identifier is as follows. First is checked of one of the names already has an identifier. If this is the case then all of the names are assigned this specific identifier. The effect when using different structures is showed in the following example. Suppose that the following DICOM XML element is parsed and indexed:

```
<Attribute Group="0018" Element="1151" Name="Patient’s name">
```

This has the result that the tag names ‘00181151’ and ‘patientsname’ are used in the index for the content that is contained in this tag. These tag names are assigned the same identifier. Next we encounter a fragment of another XML document of a different structure. Suppose this has the following format:

```
<Patientname Element="1151" Group="0018">
```

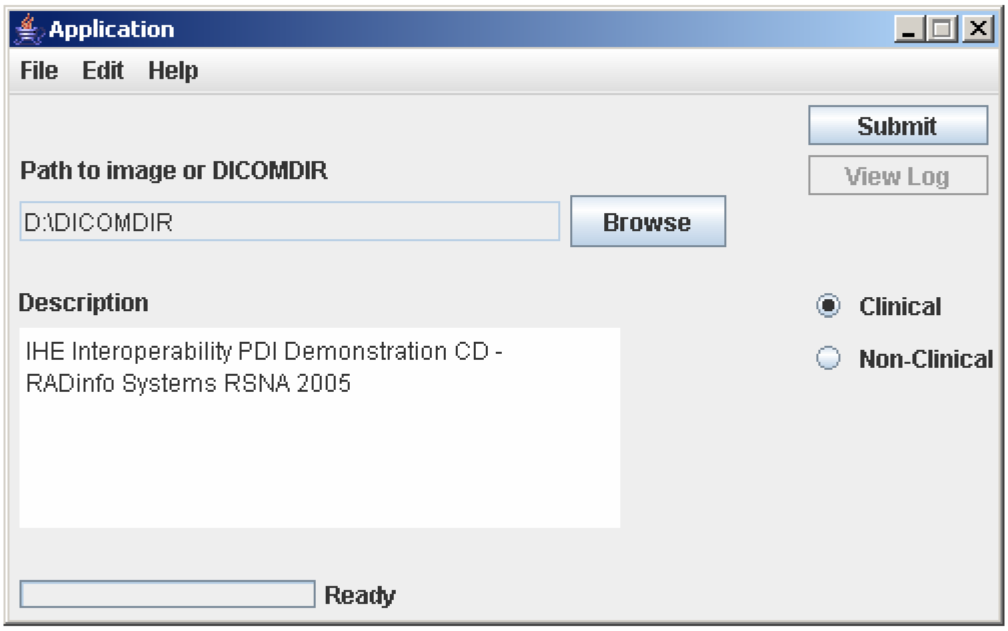
Parsing this element leads to the tag names ‘patientname’ and ‘00181151’. However the tag ‘001811511’ already has been assigned an identifier. The tag ‘00181151’ maintains this identifier and ‘patientname’ is also assigned this identifier. So after indexing both fragments the tag names ‘patientname’, ‘patientsname’ and ‘00181151’ are each other synonyms because they have the same identifiers. The connector between different names will almost always be the DICOM attribute tag, in this case (0018,1151), because like we stated in section 2 this unique identifier will always belong to the same tag.
The effect of indexing this example is that all of the resulting tag names lead to the same result when they are used in a query and that the content of both fragments is taken into account during retrieval. These aspects improve the retrieval capabilities of the application.
The described scenario seems to imply that the XML generator tool(s) that are used should make consistent use of names and identifiers, so that resulting tag synonyms are indeed synonym. We found out that this is a reasonable assumption. Still we have a short look at what the consequences are if this were not the case. Suppose that in an XML document a different DICOM tag is used for a particular name. Then it is possible that two completely different DICOM tags are assigned the same tag identifier. This means that the result of a query can contain (more) irrelevant results, although they match the query. However, without using this method this also would have been the case, only now the group of irrelevant documents that match the query has grown; in other words precision drops in that case.

Concluding this section on the creation of tag names to use in the index we can state that by introducing these parse rules we have lost a little bit of flexibility. This is caused by the fact that when completely new document structures are introduced there is a possibility that the parse rules in the configuration must be updated. However this is a simply action and on the other hand this approach leads to better retrieval results and more retrieval flexibility.

11.2 User Interface

11.2.1 Input Side

This section describes the user interface that provides the functionality for a user to load DICOM objects into the system. Related to this, this section describes the different aspects of the process of loading data into the system.



Path to image or DICOMDIR: D:\DICOMDIR

Description: IHE Interoperability PDI Demonstration CD - RADinfo Systems RSNA 2005

[✓ Clinical] Non-Clinical

Figure 10: Upload GUI

In figure 10 the user interface to load DICOM data into the system is shown. It is a very simple UI. It provides three steps that a user needs to go through. They are: (1) select an image in case of a single object or a directory containing the DICOMDIR in case of a dataset, (2) enter a meta description of the object/dataset and (3) indicate whether we are dealing with clinical images or not.

After all the information has been filled in, the process of uploading the data can be started. This process consists of the following steps. First an XML document of the meta description entered by the user is created. Also the information on whether the data is clinical is contained in this document. After the creation of the meta file, the XML generator tool is called to generate the DICOM XML documents of the images in the given path. At this point all necessary information is available and can the data be uploaded to the correct destination.
One of the requirements is that the structure of a dataset must stay intact in the application. Further do we need to link the XML representations to the images that they represent after retrieval. Also, like we described in section 10.2 need the meta files to be linked to the corresponding dataset. For these reasons the destination always has the file structure that is presented in figure 11. First, in this file structure we maintain different directory structures for datasets and single objects. Further, both datasets as single objects consist of three parts: the original data, the XML representation(s) of this data and the meta XML document. By maintaining a separate directory for the original data we make sure that the structure of a dataset stays intact.

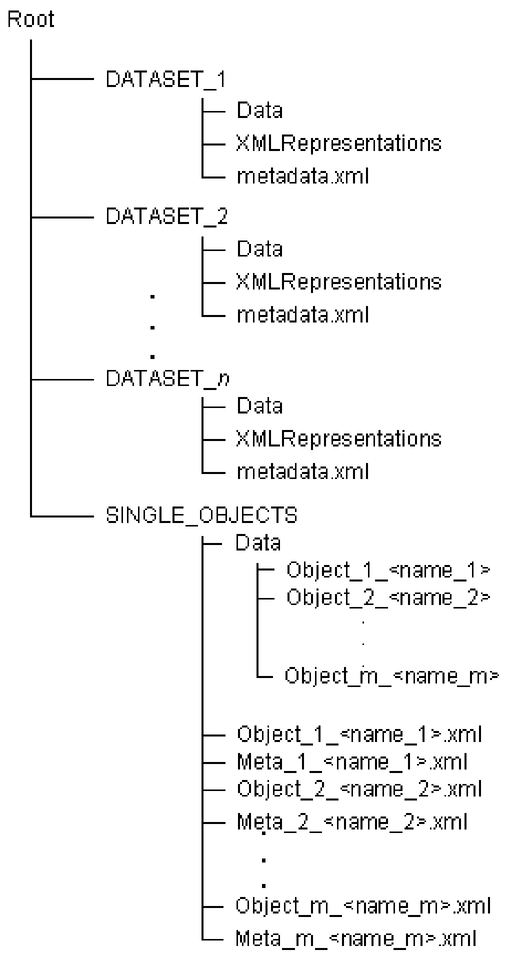


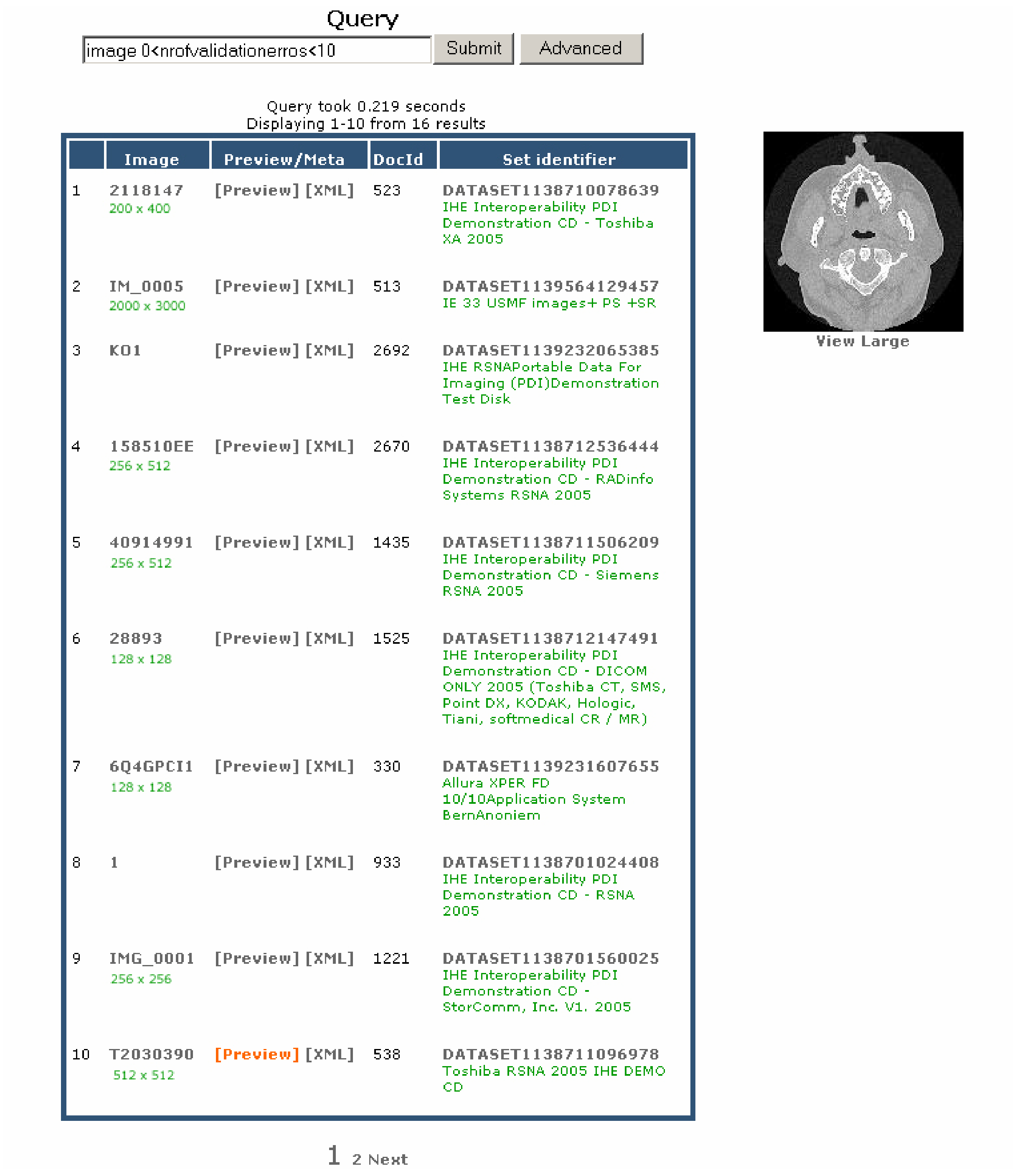
Figure 11 : Physical storage file structure

In order to quickly link the XML representation to the original DICOM images we want the filename of the representation XML document to contain the filename of the underlying DICOM object. In this case we don't have to open the files to see what the corresponding image is. This is important because when presenting the results to the user, the user must be able to quickly browse through the retrieved results. To achieve this, the XML representations are first renamed before they are copied to the destination. The small tool that renames the files is strongly related to the tool that generates the XML, because it contains the logic where to retrieve the filename of the underlying image.

After all the data has been copied to the correct destination the XML documents are ready for indexing. For each (re-)index run it is necessary to index all data again, because the index structures have to be created from scratch. For this reason indexing will probably not occur directly after the data is copied, but at fixed points in time, for example at nights.

11.2.2 Output Side

This section describes the user interface at the output side of the application. This user interface consists of a web interface that allows the user to enter his query, that shows the list of results returned for this query and that allows the user to view the details of each result. In figure 12 an example of how this interface could look like. This is an example of an interface because the interface can easily be replaced because of the architecture that was shown in figure 6. Also the web interface itself contains some replaceable components of which we will see some examples. To discuss the presentation side of the application we first deal with the queries that can be entered. Then we look at how the results are presented to the user and finally how details of the results can be viewed.



Query took 0.219 seconds
Displaying 1-10 from 16 results

	Image	Preview/Meta	DocId	Set identifier
1	2118147 200 x 400	[Preview] [XML]	523	DATASET1138710078639 IHE Interoperability PDI Demonstration CD - Toshiba XA 2005
2	IM_0005 2000 x 3000	[Preview] [XML]	513	DATASET1139564129457 IE 33 USMF images+ PS +SR
3	K01	[Preview] [XML]	2692	DATASET1139232065385 IHE RSNAPortable Data For Imaging (PDI)Demonstration Test Disk
4	158510EE 256 x 512	[Preview] [XML]	2670	DATASET1138712536444 IHE Interoperability PDI Demonstration CD - RADinfo Systems RSNA 2005
5	40914991 256 x 512	[Preview] [XML]	1435	DATASET1138711506209 IHE Interoperability PDI Demonstration CD - Siemens RSNA 2005
6	28893 128 x 128	[Preview] [XML]	1525	DATASET1138712147491 IHE Interoperability PDI Demonstration CD - DICOM ONLY 2005 (Toshiba CT, SMS, Point DX, KODAK, Hologic, Tiani, softmedical CR / MR)
7	6Q4GPCI1 128 x 128	[Preview] [XML]	330	DATASET1139231607655 Allura XPER FD 10/10Application System BernAnoniem
8	1	[Preview] [XML]	933	DATASET1138701024408 IHE Interoperability PDI Demonstration CD - RSNA 2005
9	IMG_0001 256 x 256	[Preview] [XML]	1221	DATASET1138701560025 IHE Interoperability PDI Demonstration CD - StorComm, Inc. V1. 2005
10	T2030390 512 x 512	[Preview] [XML]	538	DATASET1138711096978 Toshiba RSNA 2005 IHE DEMO CD

Figure 12 : DICOM retrieval web interface

When we look at the queries that can be entered to the system we see that the query possibilities with respect to the original Terrier framework are extended with the following two types of queries:

- `tag=term`
- $\texttt{term}_1\texttt{>tag>term}_2$ or $\texttt{term}_1\texttt{<tag<term}_2$

These two types of queries have the behavior that has been described earlier in this document.
The result of a query is an ordered list of results. The collection that the application uses to retrieve the results from is a collection of XML documents. In figure 12 we see that the link to these DICOM XML documents is also shown, but the most important aspect is the list of returned DICOM objects. Because of the directory structure of figure 11 that we discussed in the previous section, the link from XML document to DICOM object can easily be made. By also offering the XML document to the user,

the user can get more insight in the retrieval process compared to the situation where only the DICOM objects were returned. In the end the XML documents are determinative for the result.
The user can choose how the results are returned. These can be grouped by dataset or not. When the results are grouped by dataset this means that of each dataset only one result is shown, the one with the highest score. This functionality is useful when the collection is queried on characteristics that DICOM objects within one dataset have in common.

Of each returned DICOM object some detailed information is shown or can be showed. Besides the fact that the complete XML representation is returned, there are more sources of information. First of all a preview of a returned DICOM image can be shown. To show these previews we make use of the DCM4che library [16]. This library is able to extract and decode the image data from a DICOM image. Together with the Java Advanced Imaging (JAI) API [37] this data is converted to a JPEG stream and then send to the web interface to show to the user.
Further as is shown in figure 12 details are shown of both the single DICOM image as the dataset, in this case the dimensions of the image and the description of the dataset are shown respectively. These information is shown by modules that can be replaced easily so that the detailed information that is shown can be modified easily. An option in the future may be to show detailed information that is related to the query.
The user interface to present the application to the user offers enough possibilities to be changed to adapt to different situations. Partly because of the modular way the interface itself is set up and partly because it is placed in the total retrieval application in a modular way as well.

PART IV

12 Experiments

This section describes the experiments that we used to test the correct functioning of the application. For this purpose we return to the requirements of section 3 and, to a less extent, of section 6. In these sections we discussed and summarized the main requirements. These are also the main requirements that we have tested during the experiments. The main requirements for our application have been defined as: flexibility at input and output side, possibility to query the application on both content as structure, taking into account the user-defined meta-data in the retrieval process and finally the ordering of the results according to relevance.
Based on these requirements we have defined a number of experiments. The experiment can be divided into four categories:

- Basic functionality of the application
- Retrieval performance
- Flexibility
- Performance of the system (speed)

To describe our experiments we first start with the specification of the system that our application run on. Then we describe the collection of DICOM XML documents that have used during the tests and we describe the characteristics of the operational application and the details of the test setup. After that we describe the experiments of each of the categories. Of each experiment is explained how it was set up and the results are given.

12.1 Test Setting

Table 1 describes the characteristics of the system that we run our application on.

Platform	Windows XP
Processor	Pentium 4 1500 Mhz
Memory	1 GB
Application Server	Sun Java System Application Server Platform Edition 8.1
Java	Version 1.5 with 250 MB memory

Table 1: System characteristics

In table 2 the characteristics of the document collection are shown that we used during our tests.

Number of documents	40770
Number of datasets	76
Average XML document size	≈50 KB
Total size	2010MB

Table 2: Collection characteristics

Our test collection was a document collection of more than 2 GB of DICOM XML documents that consisted of more than 40.000 separate documents. This collection was created by uploading various CD’s that are available. Besides this we downloaded some DICOM datasets from [38] to create more diversion in the test collection.

In table 3 the characteristics of the application are given as it was operational on the given document collection.

Number of comparable postings	4.50 million
Number of non comparable postings	4.06 million
Total index size	429 MB

Table 3: Operational application characteristics

The number of postings that are mentioned in the table 3 are the number of pointers to documents in both the inverted indexes.

12.2 Test setup

Our basic test setup was an application that was up and running for two months. A group of about ten users tested the functionality. Four of them also had permission to load data into the system. This was useful while these users could directly try to retrieve the content that was uploaded by themselves. The queries of all users during these two months have been recorded. Not only the queries that have been entered into the application were recorded, but also the query time, the parsed version of the query and the number of returned results.

12.2.1 Basic functionality

Here we deal with the basic application functionality. First we start with the requirement that the application must be able to deal with all types of DICOM objects that are available. For loading the DICOM objects into the application we see that is dependent on the XML generator tool that is used. DVT, the tool that we used was able to create XML representations of all DICOM objects in our test collection. We also used DCMTk [15], Dcm4che [16] and Pixelmed [17], which we discuss later in this section, but these tools all had problems to generate an XML representation of some of the DICOM objects in the collection.
For the previewing of the images something similar holds. This is dependent on the performance of the dcm4che library that was used to present the images to the user. The library and therefore the application was able to show a preview of all DICOM objects that contained image data.
Finally, we formulated the requirement that the original data must stay intact. This was no problem. All uploaded data could be retrieved in the original form.

12.2.2 Retrieval performance

The experiments with respect to the retrieval functionality can be further divided into:

- Overall retrieval performance
- Meta-info in retrieval process

Retrieval performance
For the retrieval performance of the application we can make a distinction between the retrieval functionality of the application that originates from the original Terrier framework and the added data retrieval functionality. First we look at this basic functionality; the returned results and the effects of the different weighting models. Then we look at the data retrieval functionality, we look at the returned results whether they satisfy the conditions or not.

The situation where users uploaded data and afterwards tried to retrieve that data is a situation where it is certain that there are results to a query. In that case at least something must be returned. All users were able to retrieve the required results.
We used the queries that were recorded from the users to evaluate the relevance performance of the application and especially the ordering of the results. The ordering of the returned results was according the requirements. Irrelevant results were ranked in the lower parts of the result list.
Also did we use these queries to evaluate the effect on the retrieval of the different weighting models. We discovered not many differences between the different weighting models. With the different models a specific result could differ 1 or 2 places in the ranking, but none of the models placed an irrelevant result in the top of the results. Further performed all models the same, however there were two exceptions. The BB2 model of the Terrier framework had problems with queries that contained

terms that occurred only once in the complete collection. In that case no results were returned. The IFB2 model had problems with queries that contained terms that occurred relatively often in the collection. Also in this case no results were returned.

Data retrieval queries
To test the data retrieval performance of the application we have taken a 'normal' query as the basis, that is query without data retrieval parts. Of this query we save the results. Next we have added conditions to the query. We did this in such a way that these conditions split up the results into disjunctive sets. For example we first add the condition `0<rows<500` to the query and execute it. Then we replace the condition by `501<rows<5000` (based on the knowledge that objects with more than 5000 rows don't exist) and also execute this query. Then we manually inspect the different returned sets.
In this experiments we see that the different sets indeed satisfy the conditions and that the different assumed disjunctive sets are indeed disjunctive and together make up the complete set of results.

Meta info in the retrieval process
One of the requirements was that the information contained in the meta documents must be taken into account during retrieval. In section 10.2 we discussed our approach to do this. The factor that is important in this process is the multiplier to apply to the score of the meta documents. During the experiments we found that a meta multiplier of 0.6 resulted in the best balance between the weight of information in the DICOM objects and the information in the meta description. Increasing this value meant that relatively more weight was given to information contained in the meta description and decreasing it meant that more weight was given to information contained in the DICOM objects.

12.2.3 Flexibility

To test the application with respect to the flexibility of the document structure we have made use of different XML generator tools. These are the tools that were already mentioned in section 6 were we discussed some alternatives for DVT to generate DICOM XML. The tools/libraries that we have used are:

- DCMTk [15]
- Dcm4che [16]
- Pixelmed [17]

We run these tools on our collection of DICOM images in order to create XML representations of the images. This was possible because we stored an exact copy of the dataset as it was uploaded.
All these libraries have the possibility to generate an XML document of a DICOM object. However, they cannot, like DVT, read a DICOMDIR and then generate a XML document of all the DICOM objects contained in the set of that DICOMDIR. They were however able to generate an XML representation of the DICOMDIR only. We solved this problem by implementing a wrapper for these tools that reads the information contained in the XML representation of the DICOMDIR and then sequentially calls the tool to create a representation for each object in the DICOMDIR.
The results for the collections generated with the other tools were basically the same as for the collection generated with DVT. The were some small differences that were caused by the fact that these tools could not generate a representation of some of the objects.

12.2.4 Performance

We have tested the performance of the system. For the total indexing process the time it took to index the complete collection was 110 minutes. This means an average index time of 0.16 seconds per document. The major part of the indexing time is on account of the parsing and building up the direct index of the documents, about 65%.
Next we deal with the retrieval performance. Performance retrieval here means the time it takes to execute queries and return results. The retrieval performance of the system can only be measured after the threshold has been determined for which terms with a frequency above that threshold are placed in the inverted term-tag index. We found that for the performance on the queries that contained both terms as tags, this threshold had the best results for the values 1000 and 5000 for the comparable terms and non-comparable terms respectively. Decreasing this value means that the

index grows although the performance gain is very little. Increasing the value means that the retrieval performance drops.
With this values the application had good performance characteristics. All types of queries could be executed within the order of a few seconds.

PART V

13 Discussion and Conclusion

This section will conclude our report about our DICOM retrieval project. To do so we will first have a section where we will discuss the results of the experiments of the previous section. Further will we have an overall evaluation of the project. Finally we have an outlook to future work.

13.1 Results Discussion

Here we discuss the results of the previous section.
For the results of the basic functionality we can simply state that the requirements are met.
When we look at the results of the retrieval performance we notice the following aspects. Irrelevant documents are not highly ranked. The differences between the different weighting models are little. The only exceptions are the weighting models BB2 and InL2. The problems that these models suffered is caused by the formulas that they use. The BB2 weighting model had problems when only one document in the collection contained the term. This occurs for example when a unique identifier is used. In this model there is a calculation of a so called Stirling power of the total term frequency (*n*) and the total frequency minus the normalized frequency of the current document (*m*). A part of the Stirling power formula contains $\log(\frac{n}{m})$. Because the document is the only document that contains the term, *m* will be almost zero. When *m* is positive there is no problem, but when *m* is negative then it leads to a result that is not a number. *M* can become negative because the normalized term frequency is sometimes slightly greater than the original one.
The IFB2 weighting models gives no result for terms that occur relatively often. This model contains the calculation of a logarithm of the total number of documents divided by the total term frequency. When the total term frequency is higher than the total number of documents this will lead to a negative score. The rest of the calculations in this models are all multiplications so the total score is also negative in that case. So also in the case that such a keyword is not the most determinative term in a query it has an influence on the total score for a query.
For the results of the data retrieval queries we are interested in whether the conditions that these queries define are satisfied. We can say that the results are satisfying because the returned results to these queries satisfy the imposed conditions.
As an overall conclusion we can state that the overall retrieval capabilities of the system are satisfying. We must however note that because of the small differences between the weighting models - this also includes the differences between the DFR models and non-DFR models - that the combination of our application and the used collection has not fully utilized the advanced weighting mechanism of the Terrier framework with its divergence from randomness model.

The flexibility experiments turned out that the application is flexible with respect to the XML document structure and therefore indirectly also with respect to the structure of the DICOM objects. We also saw that DVT as the current XML generator tool gave the best results, because it was able to process all different DICOM objects and because it provides extra validation information. For the requirement for the flexibility in the search functionality we can say that this is guaranteed by the fact that all available information is offered to use in retrieval.
We can conclude that with this application we have created a flexible solution for the DICOM retrieval problem with respect to the future.

The performance experiments turned out that a threshold for the term frequency for terms to be placed in the inverted term-tag index was most ideal for the values 1000 and 5000 for the non-comparable index and the comparable index respectively. The differences between these values is caused by the way the different indexes are used. Both indexes can be used to look up combinations of terms and tags. The performance effect for the comparable index is stronger because this index is used to lookup ranges of values and therefore more combinations of tags and terms are looked up in one query. That is the reason that the threshold for this index is lower.
With the use of these inverted term-tag index and the given threshold, we see that the execution time for all queries is within a few seconds and therefore according to the requirements.

We saw an index time for our test collection of 110 minutes. We don't place verdict on this number because it is an offline issue, i.e. is has no effect on the functionality of the application when it is up and running.
When we look at the characteristics of the index we see that the total size of the index is about 21% of the size of the XML collection. Although we did not put tight restrictions on the size of the index, when we compare this value with a comparable project like Lucene [29] we see that also this project mentions an a index size of about 20% in its project features. This gives an indication that the size of the index of our application is not unreasonably high.

13.2 Project Evaluation

In this section we make a comparison with the results of this project and the start situation. The start situation is that in the process of testing the software of medical device sometimes has to searched through a collection of available CD's and DVD's to find a data set that will cover the functionality of the software under testing. With the current application we can state that in the new situation there is a gain in time for the search for suitable images, which can be regarded as proven. This decrease in search time is even expected to grow, because used datasets that were still found on a CD can directly be added to the online collection of DICOM images.
Further, besides the gain in time also an improvement in the test quality can be expected. However, this is an assumption and can not regarded to be proven. Because now it is easier to find suitable images, one can argue that the chance that more suitable images, i.e. images that cover a larger part of the software, are found is greater.

13.3 Outlook

In this section we will have an outlook to future work what can be done in this project.
First of the all the possibility of connecting the application to a PACS can be investigated. This can be a comparable approach as [6] that has been discussed in section 4 where the PACS served as the basic storing device for a CBIR system. By connecting the application to a PACS, users can also benefit from the extra functionality that a PACS offers.
Further can the effect of more structured reports in the collection can be investigated. At this moment only a small part of the collection consisted of structured documents, the SR's. SR's are expected to become of greater importance in general and therefore also in the testing department. The number of available SR's will increase then. The content of these DICOM objects is different compared to the images in that they are more text rich. Especially interesting is the effect on the different weighting models.
In this report we mentioned that different DICOM objects in one dataset can contain information that they have in common. It can be investigated to find a mechanism to index this information only once and look at what the effect is. To do so first an approach has to be implemented that can detect information blocks that DICOM objects in the dataset have in common. Next a way has to be found to process these information blocks. A possibility is to put this information the meta file. The purpose of this meta file is to contain information that is applicable for all objects in a dataset so it should be a suitable solution.
In this report we discussed that for the data retrieval queries we make use of one generic datatype. A consequence of this was that some data can not be correctly queried because that the values in this data are of a datatype that require a specific ordering. Maybe in future work the use of more different datatypes can be investigated or a mechanism where datatypes can be defined.
Finally, in the context of the proposed future work the possibility can be investigated to base our application on the Lucene project instead of Terrier. Maybe with the needed changes Lucene will become a better candidate.

References

1. Lucy-Bouler, T. and D. Morgenstern, *Is Digital Medicine a Standards Nightmare?* MISQ Special Issue Workshop: Standard Making: A Critical Research Frontier for Information Systems, 2003(151): p. 354-360.

2. *Digital Imaging and Communication in Medicine (DICOM)*. [Webpage] 2004 April 2006 [cited February 2006]; 3.0:[Available from: http://medical.nema.org/.

3. Bidgood, W.D., et al., *Understanding and Using DICOM, the Data Interchange Standard for Biomedical Imaging.* Journal of the American Medical Informatics Association, 1997. **4**(3): p. 199-212.

4. Horii, S., *Part Four: A Nontechnical Introduction to DICOM.* RadioGraphics, 1997. **17**(5): p. 1297-1310.

5. Oosterwijk, H. and P. Gihring, *DICOM Basics*. 2 ed. 2000: Otech Inc./Cap Gemini Ernst & Young.

6. Lehmann, T.M., B.B. Wein, and H. Greenspan. *Integration of Content-based Image Retrieval to Picture Archiving and Communication Systems*. in *Proceedings of Medical Informatics Europe (MIE 2003)*. 2003. Amsterdam: IOS Press p.

7. Mohan, A. *DICOM File Indexing Plug-in*. [Webpage] 2005 2006 [cited 2006 February]; Available from: http://desktop.google.com/plugins/i/dicom.html.

8. Long, L.R., et al. *WebMIRS - Web-based medical information retrieval system*. in *Proceedings of SPIE Storage and Retrieval for Image and Video Databases IV*. 1998. San Jose p. 392-403.

9. Lowe, H.J., et al., *Automated semantic indexing of imaging reports to support retrieval of medical images in the multimedia electronical medical record.* Methods of Information in Medicine, 1999. **38**(4-5): p. 303-307.

10. Baeza-Yates, R. and B. Ribeiro-Neto, *Modern Information Retrieval*. 1999: Addison-Wesley.

11. Rijsbergen, C.J.v., *Information Retrieval*. 1979, Butterworth-Heinemann. p. 17.

12. Larson, R.R. *Geographic information retrieval and spatial browsing*. in *Geographic Information System and Libraries:Patrons, Maps and Spatial Information*. 1996 p. 81-123.

13. Torhola, M.J. and G. Amati, *Medical Imaging Data Representation with DICOM XML*, in *EuroPACSMIR 2004 in the enlarged Europe*. 2004, Atostek Ltd: Trieste.

14. DVTk. *DICOM Validation Toolkit (DVTk)*. [Webpage] 2005 2006 [cited November 2005]; Available from: http://www.dvtk.org.

15. Offis. *DCMTK - DICOM Toolkit*. [Webpage] 2005 December 2005 [cited 2006 February]; Available from: http://dicom.offis.de/dcmtk.php.en.

16. *dcm4che, a DICOM Implementation in JAVA*. [Webpage] May 2006 [cited 2006 February]; Available from: http://dcm4che.sourceforge.net.

17. Clunie, D. *PixelMed Java DICOM Toolkit*. [Webpage] 2005 January 2006 [cited 2006 February]; Available from: http://www.pixelmed.com.

18. Ounis, I., et al. *Terrier Information Retrieval Platform*. in *Proceedings of the 27th European Conference on Information Retrieval (ECIR 05)*. 2005. Santiago de Compostela, Spain p. 517-519.

19. *Terrier Terabyte Retriever*. [Webpage] 2005 2005 [cited 2006 February]; Available from: http://ir.dcs.gla.ac.uk/terrier/.

20. Amati, G. and C.J.v. Rijsbergen, *Probabilistic models of information retrieval based on the divergence from randomness.* ACM Transactions on Information Systems (TOIS), 2002. **20**(4): p. 357-389.

21. Chamberlin, D., et al., *Mapping between XML and Relational Data*, in *XQuery from the Experts: A Guide to the W3C XML Query Language*. 2003, Addison Wesley.

22. Bourret, R. *Going Native: Making the Case for XML Databases*. [Webpage] 2005 March 2005 [cited 2006 February]; Available from: http://www.xml.com/pub/a/2005/03/30/native.html.

23. Bourret, R. *XML and Databases*. [Webpage] 2005 September 2005 [cited 2006 February]; Available from: http://www.rpbourret.com/xml/XMLAndDatabases.htm.

24. Staken, K. *Introduction to Native XML Databases*. [Webdocument] 2001 2006 [cited february 2006]; Available from: http://www.xml.com/pub/a/2001/10/31/nativexmldb.html.

25. Meier, W. *eXist: An Open Software Native XML Database*. in *Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems*. 2002: Springer-Verlag p. 169-183.

26. Fuhr, N. and K. Großjohann, *XIRQL: An XML query language based on information retrieval concepts.* ACM Transactions on Information Systems (TOIS), 2004. **22**(2): p. 313-356.

27. Fuhr, N., N. Gövert, and K. Großjohann. *HyREX: hyper-media retrieval engine for XML*. in *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*. 2002. Tampere, Finland: ACM Press p. 449-449.

28. Grabs, T. and H.-J. Schek. *Generating vector spaces on-the-fly for flexible XML retrieval*. in *Proceedings of XML and IR Workshop - 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2002. Tampere, Finland: ACM Press p. 4-13.

29. The Apache Software Foundation. *Apache Lucene*. [Webpage] 2 March 2006 [cited October 2005]; Available from: http://lucene.apache.org/java/docs/.

30. Reiss, K. *Indexing XML Documents: A Hybrid Approach*. 2003 May 2003 [cited February 2006]; Available from: http://www.isrl.uiuc.edu/~kmreiss/papers/xmlindex.pdf.

31. Egnor, D. and R. Lord. *Structured Information Retrieval using XML*. in *Proceedings of the 1st XML and Information Retrieval Workshop - 23th Annual International ACM SIGIR Conference on Research and Developement in Information Retrieval*. 2000. Athens, Greece p.

32. Liu, S., Q. Zou, and W.W. Chu. *Configurable indexing and ranking for XML information retrieval*. in *Proceedings of the 27th Annual international ACM SIGIR Conference on Research and Development in Information Retrieval*. 2004. Sheffield, United Kingdom: ACM Press, New York p. 88-95.

33. Plachouras, V., B. He, and I. Ounis. *University of Glasgow at TREC2004: Experiments in Web, Robust and Terabyte tracks with Terrier*. in *Proceedings of the 13th Text REtrieval Conference (TREC2004)*. 2004. Gaithersburg, MD, USA p. 253--259.

34. Kotsakis, E. *Structured information retrieval in XML documents*. in *Proceedings of the 2002 ACM symposium on Applied computing*. 2002. Madrid, Spain: ACM Press p. 663-667.

35. Bremer, J. and M. Gertz. *XQuery/IR: Integrating XML document and data retrieval*. in *Proceedings of the 5th International Workshop on the Web and Databases (WebDB)*. 2002 p. 1-6.

36. *Java API for XML Processing (JAXP)*. [Webpage] 2005 January 2006 [cited 2006 February]; Available from: http://java.sun.com.

37. *Java Advanced Imaging (JAI) API*. [Webpage] 2005 May 2006 [cited 2006 February]; Available from: http://java.sun.com/products/java-media/jai/.

38. *DICOM sample image sets*. [Webpage] 2005 November 2005 [cited 2006 February]; Available from: http://149.142.216.30/DICOM_FILES/.

Appendix A
Functional Requirements Specification