Semantic Querying of DICOM Structured Reports using the QLever Graph Database

Adem Aydin

Bachelor's Thesis

submitted to

Albert-Ludwigs-Universität Freiburg

Supervisor Prof. Dr. Hannah Bast
Faculty of Engineering
Department of Computer Science
Chair of Algorithms and Data Structures

January 21, 2026

University of Freiburg
Faculty of Engineering
Department of Computer Science
Chair of Algorithms and Data Structures

Author

Adem Aydin

Matriculation Number: 4330810

Examiner

Prof. Dr. Hannah Bast

Faculty of Engineering

Chair of Algorithms and Data Structures

Supervisor

Prof. Dr. Hannah Bast

Faculty of Engineering

Chair of Algorithms and Data Structures

Declaration

I hereby declare that I am the sole author and composer of this thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.

I hereby also declare that my thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date

Signature

Abstract

DICOM Structured Reports, such as the standardized Radiation Dose Structured Report (RDSR), are an invaluable source of information for radiation dose monitoring. Dose Management Systems (DMS) such as Bayer Radimetrics® and GE DoseWatch provide commercial solutions to assist medical practitioners, radiographers, and medical physics experts in ensuring that radiation doses remain, on average, below national Diagnostic Reference Levels (DRLs). However, in pursuit of serving compliance needs, advanced querying capabilities are often overlooked; non-radiation Structured Reports such as quantitative imaging biomarkers are entirely disregarded. To address this problem, we introduce *dicom2rdf*, a data pipeline that processes millions of DICOM Structured Reports and provides a search interface not only for radiation doses, but for *any* Structured Report. The main idea is modeling the bare hierarchical structure of DICOM objects as a knowledge graph with the Resource Description Framework (RDF), but additionally transforming the resulting knowledge graph using an imposed set of rules. When paired with QLever, a high performance graph database, dicom2rdf enables SPARQL queries that operate on DICOM's "data element" level and make the data of arbitrary Structured Reports accessible to medical professionals who want to obtain data that go beyond serving compliance needs.

Contents

1 Introduction 2
1.1 Motivation 3
1.2 Contribution 5
1.3 Structure of this Thesis 5

2 Background 6
2.1 Resource Description Framework 6
2.1.1 Reification 7
2.1.2 RDF Turtle 8
2.2 SPARQL and QLever 9
2.2.1 SELECT Query 10
2.2.2 CONSTRUCT Query 11
2.3 DICOM 11
2.3.1 Data Element, Data Set, and DICOM file 11
2.3.2 Content Sequence 13
2.3.3 Structured Reporting Templates 13
2.3.4 Controlled Terminologies 15

3 Related Work 16
3.1 Metadata quality knowledge graphs 16
3.2 DICOM-to-RDF converters 16

4 *dicom2rdf*: Turning DICOM files into a Knowledge Graph 18
4.1 *convert*: Turning DICOM SR into intermediate RDF 18
4.1.1 From a DICOM Data Element to an RDF triple 18
4.1.2 Creating and writing triples to disk 19
4.1.3 DICOM file traversal 21
4.2 *construct*: Transforming RDF for better queries 25
4.2.1 Why *construct*: disjoint collection of top-level triples 25
4.2.2 Why *construct*: too many required hops to reach concepts and measurements 26
4.2.3 Using SPARQL CONSTRUCT queries to form new triples 29
4.3 Optimizations: Scaling from Thousands to Millions of DICOM SRs 31
4.3.1 Parallelize conversion 31
4.3.2 Spending CPU time to cope with I/O boundedness 31
4.3.3 Progress Logging with multiple workers 31

4.3.4 Starting multiple smaller QLever instances . . . 32
4.4 Orchestrating the *dicom2rdf* pipeline . . . 32

5 Evaluation . . . 33
5.1 Performance Evaluation . . . 33
5.1.1 Setup . . . 33
5.1.2 Methodology . . . 35
5.1.3 Benchmark: *convert* vs. *dcm2rdf* . . . 35
5.1.4 Preparation: *dcm2rdf* . . . 36
5.2 *dicom2rdf* in a clinical context . . . 37
5.2.1 Dose Management . . . 37

6 Conclusion and Future Work . . . 40
6.1 Conclusion . . . 40
6.2 Future Work . . . 40
6.2.1 Rethink CONSTRUCT strategy . . . 40
6.2.2 Take containers within non-container Value Types into account . . . 41
6.2.3 Capture values of remaining Value Types . . . 41
6.2.4 Auto-tune parameters such as compression level . . . 42

Acknowledgments . . . 43

Bibliography . . . 44

Glossary . . . 46

A Full SPARQL Queries . . . 47

List of Figures

Figure 1 Most frequent acquisition protocols 3

Figure 2 Issues of systems specializing in standardized SR templates 4

Figure 3 a is related to b, and b is related to a 6

Figure 4 A rather bold statement about Earth's shape 6

Figure 5 Using IRIs, typed literals, and language tags to disambiguate the statement in Figure 4 7

Figure 6 An attempt at modeling Earth's population 7

Figure 7 A reified *quantity*, a pair of magnitude and unit 8

Figure 8 The statement made in Table 3 13

Figure 9 Structure of the TID-10011 CT Radiation Dose template 14

Figure 10 A series of statements about a TID-10011 CT Radiation Dose SR document . 14

Figure 11 Triple denoting "Patient's Age" from the parent (here: root) to a primitive value 23

Figure 12 Triples from the parent (here: root) to multiple Sequence Items, each with their own mediator 23

Figure 13 Five hops are required to ensure that the destination concept is a CT dose. The index and Code Meaning (0008,0104) is not strictly required. (CNCS = Concept Name Code Sequence) 28

Figure 14 CNCS has been eliminated and its information encoded elsewhere 29

Figure 15 Intermediate node has been contracted and the index information moved to the concept, if needed 29

Figure 16 *dicom2rdf* pipeline 32

Figure 17 A selection of reports 37

Figure 18 BMI-dose data correlation 38

Figure 19 Leg deformity 38

Figure 20 BMI-hip-knee angle (HKA) deviation 39

List of Tables

Table 1 A simplified representation of a Data Element 11

Table 2 A simplified representation of a Data Set with each Data Element's Attribute Name. Note how Pixel Spacing has more than one value. 12

Table 3 TEXT Content Item in a Dose Report specifying the acquisition protocol, i.e. a predefined set of parameters for a CT scan. 13

Table 4 Test dataset obtained from NCI 34

Table 5 Modality breakdown 34

Table 6 Performance comparison dcm2rdf vs. convert 37

List of Listings

Listing 1 Earth's axial tilt measurement in RDF Turtle 8

Listing 2 A `Qleverfile` 9

Listing 3 The first row of a simple SPARQL query against our dataset in QLever, which responds in JSON 10

Listing 4 SPARQL query that retrieves all axial tilt measurements of Earth 10

Listing 5 A CONSTRUCT query that yields a single triple: `my-iri:foo my-iri:bar "23.44 °"` 11

Listing 6 Basic usage of dicom-rs 19

Listing 7 A Turtle file with two triples as they would appear when converting a DICOM file 19

Listing 8 IRI representation in Rust 20

Listing 9 A triple object, which can either be an IRI or a literal 20

Listing 10 An RDF triple in Turtle 20

Listing 11 Building a Turtle triple from smaller parts 20

Listing 12 Writing a triple to a buffer 21

Listing 13 A birds-eye view of the recursive DICOM file conversion process 22

Listing 14 SPARQL query that retrieves a single SR with the study date, study time, patient's age, family name, and given name. 25

Listing 15 SPARQL query that retrieves all Mean $CTDI_{vol}$ measurements on the dataset produced by *convert*. 27

Listing 16 SPARQL query that retrieves all Mean $CTDI_{vol}$ measurements on the new dataset produced by *construct*. 30

Listing 17 Same as Listing 16 but with Turtle's syntactic sugar 31

Listing 18 **Dataset provided by the University Medical Center Freiburg, numbers obtained with SPARQL query** **35**

Listing 19 **Initiating the dicom2rdf benchmark for LUNG1-001** **36**

Listing 20 **Dockerfile used to benchmark *dcm2rdf*** **36**

Listing 21 **Initiating the dcm2rdf benchmark for LUNG1-001** **36**

Listing 22 ..... **47**

CHAPTER 1

Introduction

> *What are the dose level differences for the protocol "0603_ Thorax_pulmart" on a SOMATOM Edge Plus scanner vs. a SOMATOM Force scanner? Does one device consistently yield lower doses? What about different protocols on the same device? How differently do they perform on a patient with a BMI lower than 20 vs. a patient with BMI higher than 25?*

These are the types of questions a Medical Physics Expert (MPE) is occupied with. The goal is to ensure minimum radiation dose levels for a patient while maintaining an image quality that supports the diagnostic process—dose levels that are "As Low As Reasonably Achievable" (ALARA) [1].

A common medical imaging procedure, the Computed Tomography (CT) scan, involves the continuous emission of X-rays directed at a patient's body part in order to obtain a medical image. X-rays are ionizing radiation that have well-documented adverse health effects [2]. A recent study suggests that "if current radiation dosing and utilization practices continue, CT-associated cancers could eventually account for 5% of all new cancer diagnoses annually" [3].

Hospitals use a Dose Management System (DMS) to ensure that dose levels remain below national Diagnostic Reference Levels (DRL). Monitoring DRLs has become a mandatory practice since the passing of Directive 2013/59 Euratom [4], emerging a market of different DMS vendors. To ensure interoperability among vendors, the Digital Imaging and Communications in Medicine (DICOM) [5] standard ensures that devices from different vendors adhere to the same communications protocol and file format. One particular specification is the DICOM Structured Report (DICOM SR).

A DICOM SR is a form of DICOM object: a collection of "Data Elements", i.e. key-value pairs, that can be nested. The Radiation Dose Structured Report (RDSR) is a more specific SR, where the standard describes exactly which Data Elements must be present in which arrangement. It is the primary means by which devices exchange dose levels and device parameters. Therefore, RDSRs with its predictable arrangement of data, are the primary data source of DMS and make dose management possible in the first place.

1.1 Motivation

Discussions with a Lead MPE at the University Medical Center Freiburg have revealed a lack of querying capabilities when retrieving dose measurements from a DMS. For example, a study with a pre-monitoring and monitoring stage for bolus tracking may influence the Computed Tomography Dose Index ($CTDI_{vol}$) reported on the study level [6]. The capability to filter out such individual values from a RDSR, however, may not be possible and cause discrepancies between dose values reported by the CT scanner and DMS [7].

	?acquisition_protocol	?count_acquisition_protocol
1	Topogram	271,227
2	PreMonitoring	110,649
3	Monitoring	109,904
4	Thorax Abdomen Routine	88,480
5	Topogramm	87,624

Figure 1: Most frequent acquisition protocols

Figure 1 shows that four out of five of the most frequent acquisition protocols are non-diagnostic, i.e. they serve calibration functions. If a DMS were to incorporate the doses of non-diagnostic series, it may impact the accuracy of reported numbers for a specific diagnostic series. When comparing dose levels of diagnostic series against national DRL, following established methodology, non-diagnostic series should be excluded [8], [9].

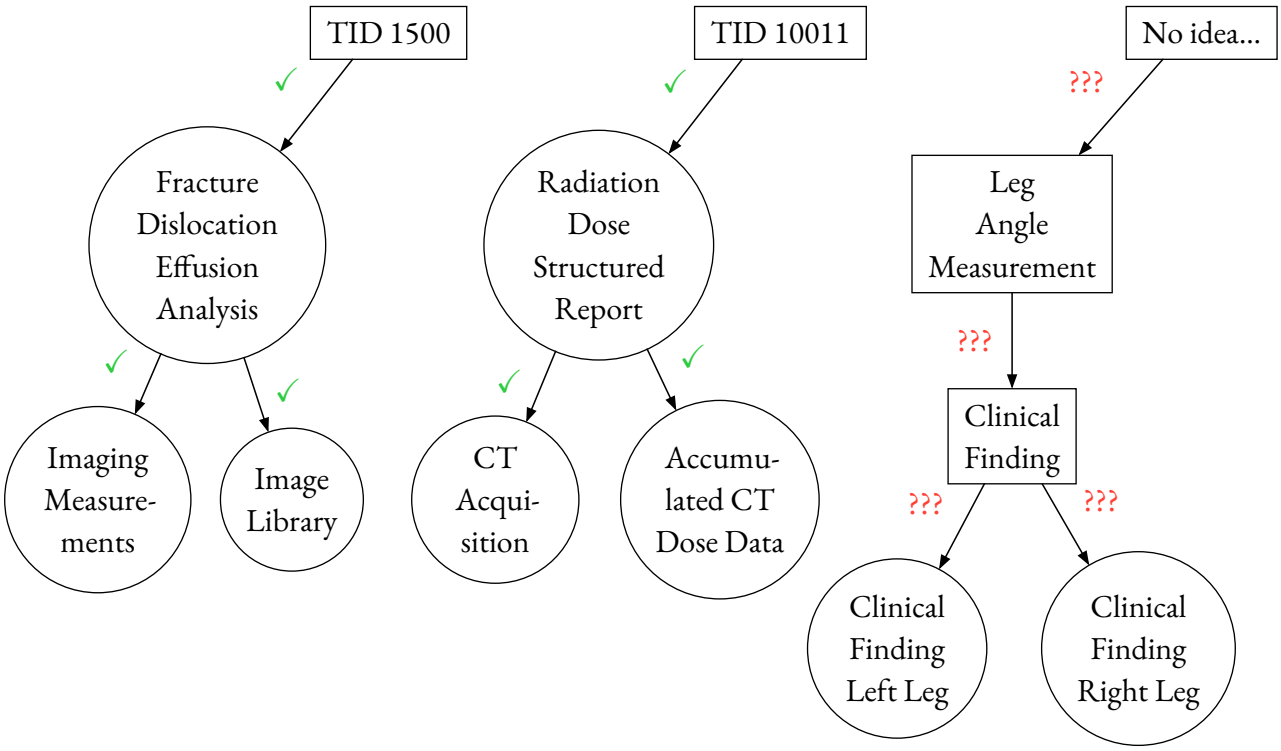

Figure 2: Issues of systems specializing in standardized SR templates

On the other side of the spectrum are DICOM SRs that are not dose-related, but contain results of so-called "quantitative imaging biomarkers"[10]: DICOM SRs that encode the interpretation of a biologic feature within an image region. External vendors leverage the DICOM standard to benefit from its widespread use, but may not use a standardized DICOM SR template. Therefore, analysis of outcome-based SR is not as clear as analysis of RDSRs, which do adhere to a template. A vendor that specializes in template-based SRs, let alone only RDSRs, simply cannot deal with the heterogeneity of any SR (Figure 2). Therefore, the problem we want to solve is clear:

Given a large collection of Structured Reports, how can we make each of its individual Data Elements queryable, regardless of whether they adhere to a SR template?

1.2 Contribution

In collaboration with the Department of Radiology at the University Medical Center Freiburg, this thesis examines a collection of over 7 million DICOM SR documents and explores efficient ways to query the data. A particular focus is the use of knowledge graphs as a data modeling tool. As a result, we introduce ***dicom2rdf***, a data pipeline that is capable of processing millions of DICOM SR files to RDF, makes the data searchable with SPARQL, and is powered by QLever, a high performance graph database.

In the first pass, our solution does what other RDF converters do and translates the bare hierarchical structure of a DICOM file to RDF triples. However, a second pass transforms the knowledge graph further with a set of dynamically generated SPARQL CONSTRUCT queries. The resulting RDF dataset has a reduced triple count and makes use of the coded concepts embedded in a DICOM file. The resulting queries follow a semantic series of statements that are easy to understand. This allows running queries in datasets that are much larger than previously possible. The pipeline uses standard containerization technologies and makes it easy to deploy without requiring a lot of dependencies.

Initial results show that the combination of dicom2rdf, QLever and SPARQL allow for highly expressive, swiftly executed queries that go beyond radiation dose monitoring. The project is open-source and available on GitHub[1].

1.3 Structure of this Thesis

Section 2 provides the necessary background knowledge, especially knowledge pertaining to medical imaging, to understand the thesis.

Equipped with this knowledge, Section 3 highlights related work done in the field of DICOM Structured Reports and knowledge graphs.

Section 4 introduces dicom2rdf and its two stages *convert* and *construct*, where both the general approach and implementation details are discussed.

In Section 5, we observe both the performance characteristics and practical utility of dicom2rdf.

[1] https://github.com/ad-freiburg/dicom2rdf

CHAPTER 2

Background

This chapter provides the necessary background knowledge to understand the concepts we present in this thesis. Section 2.1 provides an introduction to RDF and how it is used to model information as a knowledge graph. Section 2 introduces SPARQL and QLever as a means to extract knowledge from a knowledge graph. Section 2.3 provides a high level overview of the DICOM standard, especially the parts that are relevant to us.

2.1 Resource Description Framework

The *Resource Description Framework* (RDF) is a standard to describe and exchange relationships of objects as a graph.

A graph is a structure that defines a set of objects and whether or not they are related.

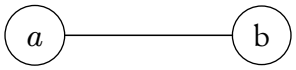

Figure 3: a is related to b, and b is related to a

A graph can be directed to specify the direction of a relation, and labeled to assign the relation some semantic meaning.

The fundamental concept of RDF is that of a *triple*, consisting of a *subject*, *predicate*, and *object*. By populating the subject, predicate, and object with concrete concepts, RDF allows us to make statements about the world.

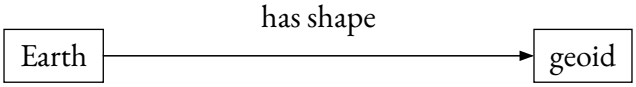

Figure 4: A rather bold statement about Earth's shape

In Figure 4, the subject is *Earth*, the predicate is *has shape*, and the object is *geoid*.

However, it is not clear whether we refer to Earth as a *planet*, Vangelis' first solo album titled *"Earth"*, or Earth as in *soil*. The RDF standard defines ways to disambiguate the meaning of concepts.

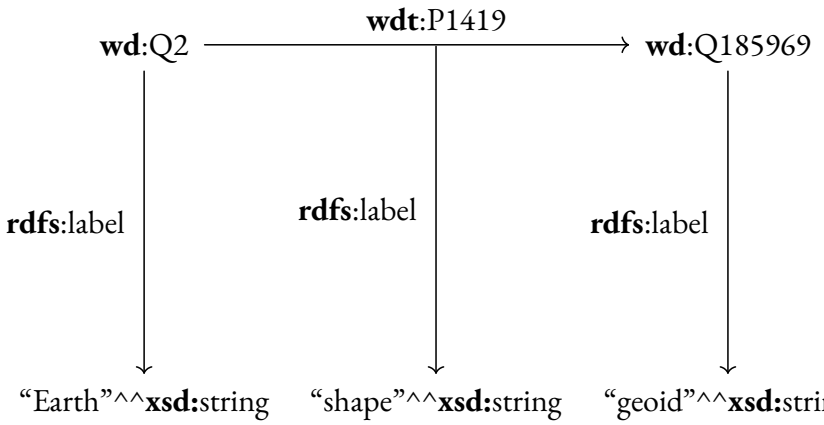

Figure 5: Using IRIs, typed literals, and language tags to disambiguate the statement in Figure 4

In Figure 5, we disambiguate the concept "Earth" by referring to its *Internationalized Resource Identifier* (IRI) within an established ontology–Wikidata. An IRI looks like URL, but allows the use of more characters. The full IRI of *Earth* is <http://www.wikidata.org/entity/Q2>. By specifying an *IRI prefix* such as wd: http://www.wikidata.org/entity/, we can abbreviate the IRI to be more compact. Notably, an IRI does *not* have to resolve to a reachable web address, or serve content related to the IRI–its sole purpose is to associate a concept with a unique name. A URL-like string just happens to work well.

Finally, the **rdfs:**label predicate resolves the hard-to-read IDs to human-readable strings, so-called *typed literals*. The type itself is an IRI. The default type of a literal is **xsd:**string, therefore the type specifier can be omitted.

2.1.1 Reification

Suppose we want to record Earth's axial tilt as an RDF triple. Since, at the time of writing this thesis, *axial tilt* does not exist in Wikidata as an IRI intended to be used as a predicate, we *mint* our own IRI. This means we put our own IRI out in the world and assign our own semantic meaning to concepts defined within our IRI. A possible approach is shown in Figure 6.

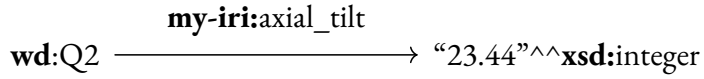

Figure 6: An attempt at modeling Earth's population

We use our newly minted IRI to describe Earth's axial tilt with an **xsd:**float, another typed literal.

This representation has at least two issues:

- It is not specified anywhere that the axial tilt is measured in degrees.
- It is only a snapshot of Earth's current axial tilt at the time of writing this thesis.

Instead, the statement we would like to make is:

> In January 21, 2026, Earth's axial tilt was recorded to be 23.44 degrees.

This statement connects five concepts: a date, the Earth, the axial tilt, a numeric measurement, and a unit.

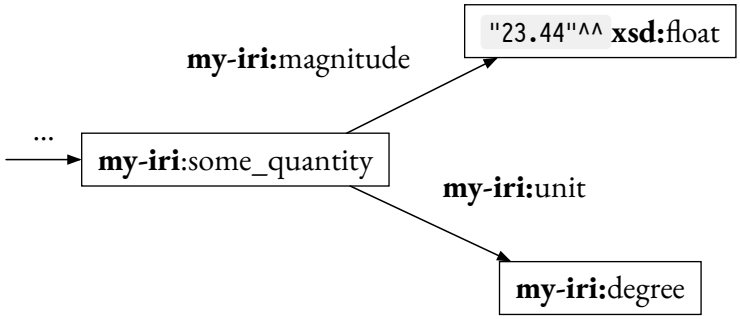

Figure 7: A reified *quantity*, a pair of magnitude and unit

One solution is *reification*: we form a series of triples using the same *mediator* in the subject, such that we can refer to that mediator in the object. Figure 7 shows one such reification. Note that the mediator is not limited to just two predicates, it can be enriched with as many predicates as needed. This can be repeated indefinitely to build a model that is sufficiently expressive.

2.1.2 RDF Turtle

So far, we have only seen visual representations of RDF. In order to store and transmit RDF, we need a serialization format. A popular format is *Terse RDF Triple Language* (Turtle). The file extension of a Turtle file is `.ttl`.

```
@prefix wd: <http://www.wikidata.org/entity/> .
@prefix my-iri: <https://mydomain.com/rdf/> .

wd:Q2 my-iri:axial_tilt_measurement my-iri:some_measurement .
my-iri:some_measurement my-iri:timestamp "2025-01-10T12:34:56.789Z" .
my-iri:some_measurement my-iri:quantity my-iri:some_quantity .
my-iri:some_quantity my-iri:magnitude "23.44"^^xsd:float .
my-iri:some_quantity my-iri:unit my-iri:degree .
```

Listing 1: Earth's axial tilt measurement in RDF Turtle

In Turtle, every triple is written on its own line and ends in a period. The subject, predicate, and object are separated by a space. A literal space character must therefore be percent-encoded with %20. Prefix declarations are placed at the top.

2.2 SPARQL and QLever

We have created our own RDF dataset and now want to query it. The SPARQL Protocol and RDF Query Language (SPARQL) is the default language to query RDF. QLever is a graph database that runs such SPARQL queries. In order to use QLever, the dataset needs to be indexed first.

```
[data]
NAME              = my-dataset
DESCRIPTION       = My Dataset

[index]
INPUT_FILES     = dataset.ttl
CAT_INPUT_FILES = cat ${INPUT_FILES}
SETTINGS_JSON   = { "ascii-prefixes-only": false, "num-triples-per-batch": 100000,
"parallel": false }

[server]
PORT               = 8000
ACCESS_TOKEN       = ${data:NAME}-my-access-token
MEMORY_FOR_QUERIES = 5G
CACHE_MAX_SIZE     = 2G
TIMEOUT            = 30s

[runtime]
SYSTEM = docker
IMAGE  = docker.io/adfreiburg/qlever:latest

[ui]
UI_CONFIG = my-dataset
```

Listing 2: A `Qleverfile`

A companion tool qlever-control provides the CLI utility `qlever` that makes it easy to build an index. Given a file `dataset.ttl` with the contents of Listing 1 and so-called `Qleverfile`, we index the dataset simply by calling `qlever index`.

After the index has been created, we follow up with the `qlever start` command, which makes the server available. This enables us to run SPARQL queries via HTTP, as shown in Listing 3. But what is a SPARQL query?

```
curl -G "http://localhost:8000" \
     --data-urlencode "query=SELECT * WHERE { ?s ?p ?o }" |
     jq -r '.results.bindings[0]'
```

```
{
  "s": {
    "type": "uri",
    "value": "https://mydomain.com/rdf/some_quantity"
  },
  "p": {
    "type": "uri",
    "value": "https://mydomain.com/rdf/magnitude"
  },
  "o": {
    "datatype": "http://www.w3.org/2001/XMLSchema#decimal",
    "type": "literal",
    "value": "23.44"
  }
}
```

Listing 3: The first row of a simple SPARQL query against our dataset in QLever, which responds in JSON

2.2.1 SELECT Query

A SPARQL SELECT query extracts data from a knowledge graph. In the WHERE clause, we specify a series of triple patterns. These look like RDF triples, but allow for the placement of variables, prefixed with a question mark. Variables can be reused across triple patterns, which tells the SPARQL engine that if a triple has been found that matches one triple pattern, the value of its variables must be the same across the whole query. Finding variables that satisfy all triple patterns in a query is the key challenge solved by a graph database such as QLever.

```
PREFIX wd: <http://www.wikidata.org/entity/>

SELECT ?measurement_time ?magnitude ?unit WHERE {
    wd:Q2 my-iri:axial_tilt_measurement ?measurement .
    ?measurement my-iri:timestamp ?measurement_time .
    ?measurement my-iri:quantity ?measurement_quantity .
    ?measurement_quantity my-iri:magnitude ?magnitude .
    ?measurement_quantity my-iri:unit ?unit .
}
```

Listing 4: SPARQL query that retrieves all axial tilt measurements of Earth

Listing 4 shows an example query.

2.2.2 CONSTRUCT Query

A SPARQL CONSTRUCT query looks very similar to a SELECT query. Consider Listing 5, which consists of a triple pattern followed by a BIND operation, which creates a new variable. This variable concatenates the magnitude's numeric value with the " °" string. In the CONSTRUCT clause, we simply define new triples and refer to variables of a triple pattern.

```
CONSTRUCT {
    <my-subject> <my-predicate> ?degree_with_unit_symbol
} WHERE {
    ?s my-iri:magnitude ?o .
    BIND(CONCAT(STR(?o), " °") as ?degree_with_unit_symbol)
}
```

Listing 5: A CONSTRUCT query that yields a single triple: `my-iri:foo my-iri:bar "23.44 °"`

CONSTRUCT queries allow us to create new RDF datasets with an entirely new structure. It is a fundamental tool this thesis uses.

2.3 DICOM

This section introduces aspects of the DICOM standard necessary to understand the thesis. Due to the DICOM standard's complexity, we first introduce fundamental concepts. As we broaden the scope, we abstract away certain details of the DICOM standard.

Terms that are used in general language (e.g. *Value*) are capitalized whenever their meaning corresponds to the definition given by the DICOM standard.

2.3.1 Data Element, Data Set, and DICOM file

Data Element

A *Data Element* encodes a semantic concept and its value. For example, the patient's age is represented by the Data Element shown in Table 1. The *Tag* uniquely identifies the Data Element, the *Value Field* contains zero or more values, and the *Value Representation (VR)* specifies how to interpret each value.

Tag	Value Field	VR
(0010,1030)	"024Y"	AS

Table 1: A simplified representation of a Data Element

For example, the VR *Age String (AS)* must contain exactly four characters, the first three characters must be digits from 0-9, and the last character must be D, W, M, or Y.

A Data Element is a concrete instance of a *DICOM Attribute*. The DICOM standard specifies the Attribute with Tag (0010,0030) to define the semantic concept "Patient's Age" and provides a more detailed description.

In reality, the Data Element encodes more information than displayed, so that parsers can interpret the file without ambiguity. However, this simplified representation is sufficient for our purpose.

Data Set

A *Data Set* is a collection of Data Elements and each Data Element can be addressed by its Tag. A simplified Data Set is shown in Table 2. For clarity purposes, the table also shows the Data Element's corresponding Attribute Name and is not part of a Data Set.

Attribute Name	Tag	VR	Value Field
Patient's Name	(0010,0010)	PN	"Doe^Jane"
Patient's Age	(0010,1030)	AS	"024Y"
Pixel Spacing	(0028,0030)	DS	0.0042, 0.0123
Content Sequence	(0040,A730)	SQ	<table><tr><td>Attr...</td><td>Tag</td><td>VR</td><td>Value</td></tr><tr><td>⋮</td><td>⋮</td><td>⋮</td><td>⋮</td></tr></table>
Pixel Data	(7FE0,0010)	OB or OW	[31, 245, ...]

Table 2: A simplified representation of a Data Set with each Data Element's Attribute Name. Note how Pixel Spacing has more than one value.

The VR of a Data Element corresponds to data types commonly found in data interchange formats:
- String
- Floating-point number (32-bit or 64-bit, depending on VR)
- Binary data (bytes or 16-bit words)
- *Data Set*

As we have previously seen, string-based VRs further specify how to interpret the string. The VR *Person Name* (PN) uses a designated *component delimiter* to split the name into smaller components such as "family name" and "given name".

A notable VR is *Sequence* (SQ). A Data Element with a VR of SQ contains a sequence of *Items*, and each Item is itself a Data Set.

Since the focus of this thesis is the querying of DICOM SR documents, we ignore all Data Elements with a VR denoting binary contents, such as Pixel Data.

Finally, a DICOM file simply encapsulates a single Data Set and has the **.dcm** file extension.

2.3.2 Content Sequence

The Data Set of each DICOM file has a *Modality* Data Element. A Modality of CT denotes a CT scan, a Modality of SR a Structured Report, and so on.

A Structured Report is a special kind of DICOM file as it has a *Content Sequence*. The Content Sequence can have an arbitrarily nested structure. An Item in the Content Sequence, a *Content Item*, denotes a singular concept and specifies a corresponding value. An example is given in Table 3.

Attribute	Value
Value Type	"TEXT"
Concept Name Code Sequence	<table><tr><th>Attribute</th><th>Value</th></tr><tr><td>Code Value</td><td>"125203"</td></tr><tr><td>Coding Scheme Designator</td><td>"DCM"</td></tr><tr><td>Code Meaning</td><td>"Acquisition Protocol"</td></tr></table>
Text Value	"Thx/Abd DE ven"
Relationship Type	"CONTAINS"

Table 3: TEXT Content Item in a Dose Report specifying the acquisition protocol, i.e. a predefined set of parameters for a CT scan.

The *Value Type* specifies that this Content Item has a *Text Value* Data Element. Later on, we will see other Value Types to specify more than just text.

The *Concept Name Code Sequence* specifies the concept that this Content Item describes. Together with the *Text Value*, the Content Item makes the statement shown in Figure 8.

The Acquisition Protocol is Thx/Abx DE ven. The DCM vocabulary defines Acquisition Protocol via code 125203.

Figure 8: The statement made in Table 3

This way of describing coded concepts is commonplace within DICOM SR documents.

2.3.3 Structured Reporting Templates

The items in a Content Sequence are required to provide a coded concept, to specify its Value Type, and to have a value corresponding to that Value Type. While this provides some structure to

a DICOM SR document, it is not clear how these items should be arranged amongst themselves, especially given that a Container item allows for arbitrarily nested structures.

To address this issue, the DICOM standard provides *Structured Reporting Templates* (SR template). An SR template specifies exactly which Content Items must be present how many times, what their Relationship Type is to their parent, and which concepts they describe. If the SR template specifies the presence of a CONTAINER Content Item, it also recursively specifies how its children are structured. This ensures SR documents to be interoperable and predictable in structure.

In addition to specifying its Content Items, an SR template can also be defined in terms of other SR templates.

CT Radiation Dose Report

A widely used template is the *TID-10011: CT Radiation Dose* template and is defined in terms of other SR templates.

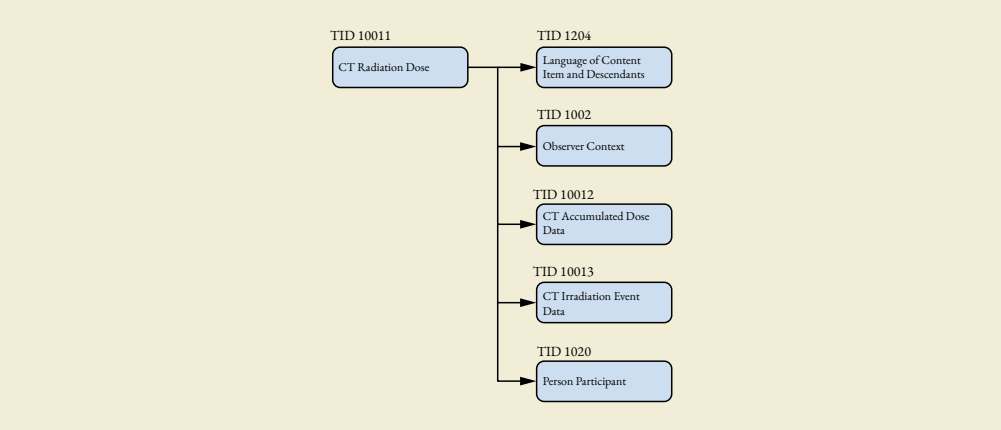

Figure 9: Structure of the TID-10011 CT Radiation Dose template

For example, suppose we are interested in documenting every $CTDI_{vol}$ measurement of a dose report. Using the information provided by the CT Radiation Dose SR Template, we can make the statements in Figure 10.

- A *CT Radiation Dose SR document* contains one or more *CT Acquisitions*
- Each *CT Acquisition* contains exactly one *CT Dose*
- Each *CT Dose* contains exactly one *Mean* $CTDI_{vol}$
- Each *Mean* $CTDI_{vol}$ holds a numeric value of unit *mGy* as defined by the *UCUM terminology*

Figure 10: A series of statements about a TID-10011 CT Radiation Dose SR document

In later chapters, we refer back to this list of assertions as it directly informs the design of our query.

2.3.4 Controlled Terminologies

DICOM SR documents make extensive use of controlled terminologies. The *DICOM Controlled Terminology Definitions* provides a list of terminologies related to medical imaging procedures such as *Acquisition Protocol*, as shown in Table 3.

Systematized Nomenclature of Medicine - Clinical Terms (SNOMED CT) is the world's most comprehensive healthcare terminology with over 350′000 terms (January 2020). In DICOM SR, it is denoted with *LN*.

Other, more specialized terminologies, have emerged over time:

- *Logical Observation Identifiers Names and Codes (LOINC)* provides a terminology surrounding medical laboratory observations and test results
- *RadLex* provides terminology for concepts in radiology
- *Unified Code for Units of Measure (UCUM)* provides terminology for expressing measurement units

Some of these terminologies have merged or became obsolete over time. As a result, it is not unusual for DICOM SR documents to contain references to terminologies that no longer exist.

CHAPTER 3

Related Work

In this chapter, we take a look at related work in the area of representing DICOM files as a knowledge graph. While previous approaches exist, we conclude that these approaches either do not fit our use-case, or produce knowledge graphs that are difficult-to-query.

3.1 Metadata quality knowledge graphs

Bejaoui et al. (2025)[11] transform DICOM metadata into a knowledge graph to assess metadata quality in imaging trials.

Metadata quality in the context of this paper refers to e.g. attribute completeness, where attribute refers to the concept of a DICOM Attribute that we have introduced in Section 2.3. The paper is limited to imaging modalities such as CT and MRT. As such, they focus on top-level attributes, which differ depending on the imaging modality. They do not, however, cover the Structured Reporting modality, where relevant attributes such as radiation dose levels are stored. Therefore, while the utilized technology—knowledge graphs—is comparable to our approach, the work serves a different purpose.

3.2 DICOM-to-RDF converters

The goals of the open source projects **dcm2rdf**[12] and **dicom2rdf**[13] are much more closely aligned with the goal of this thesis. Since both projects are employing similar conversion strategies, we focus on the newer Java-based approach, dcm2rdf, as dicom2rdf by Bonubase is based on Python 2.

The open-source project **dcm2rdf**[12] converts DICOM files to RDF by mapping tags to predicates and representing nested sequences as blank-node structures. The output stays close to the DICOM file layout: predicates mirror tag numbers and deep traversal is required to reach coded concepts. These properties make complex clinical questions hard to express and keep queries tightly coupled to DICOM's file structure, leaving a need for additional transformation before the data becomes easy to query semantically.

Our approach will consist in parts of a stage that mimics the behavior of existing projects such as dcm2rdf or dicom2rdf by Bonubase. The difference is a focus on speed and a transformation to yield triples that are easier to query.

CHAPTER 4

dicom2rdf: Turning DICOM files into a Knowledge Graph

*This chapter introduces **dicom2rdf** and the two stages it is composed of. Section 4.1 introduces **convert**, a program that takes DICOM files as input and produces RDF Turtle as output. Section 4.2 motivates why the RDF output of **convert** does not meet our needs and introduces **construct**, a program that transforms RDF into a new schema based on an imposed set of rules. Section 4.3 address several performance issues to scale from thousands to millions of DICOM files. Section 4.4 concludes the chapter with an overview of the pipeline that connects the stages together. The programs are implemented in the Rust programming language[2].*

4.1 *convert*: Turning DICOM SR into intermediate RDF

In this section, we specify how a DICOM file is processed and written as RDF to disk. Section 4.1.1 defines how each Data Element of a DICOM file is to be translated to RDF. Section 4.1.2 introduces the necessary program structures to represent an RDF triple in memory and write it to disk. Section 4.1.3 combines these insights to define the way we traverse a DICOM file and write a ***.ttl*** file to disk. The artifact of this section is the ***convert*** binary.

4.1.1 From a DICOM Data Element to an RDF triple

Before we convert a DICOM file, we must understand its contents.

In this section, we observe the contents of *dose-report.dcm*, a DICOM file that serves as a running example for the remainder of this chapter. The library *dicom-rs* is a Rust implementation of the DICOM standard and provides an idiomatic abstraction for working with a DICOM file. Listing 6 shows the basic usage of dicom-rs to open a DICOM file, read the "Modality" Data Element, and print its value. The API surface is idiomatic Rust, where functions that may fail return an `Option<T>` or `Result<T, E>` with an appropriate `Error` object.

[2] https://rust-lang.org/

```rust
fn main() -> Result<(), Box<dyn std::error::Error>> {
    let dcm_obj = dicom::object::open_file("dose-report.dcm")?;
    let modality_data_element = dcm_obj
        .get(dicom::core::Tag(0x0008, 0x0060))
        .expect("Modality should be present");
    let modality_value = modality_data_element.string()?;
    println!("{}", modality_value);
    Ok(())
}
```

```
$ cargo run
"SR"
```

Listing 6: Basic usage of dicom-rs

A DICOM file, which is a filesystem representation of a DICOM object, consists of a fixed set of top-level Data Elements. A DICOM object's Information Object Definition (IOD) specifies the Data Elements available in the object. For example, the Basic Text SR IOD [14] specifies that it consists of several modules, with Patient Module[15] being one of them. Therefore, the DICOM object contains, among others, the "Patient's Name" Data Element (0010,0010). The SOP Class UID Data Element (0008,0016), which every DICOM object has, identifies the type of DICOM object. For example, our *dose-report.dcm* example file has the SOP Class UID `1.2.840.10008.5.1.4.1.1.88.67`. The DICOM standard specifies the full list of SOP Class UIDs[16]. It is, therefore, an X-Ray Radiation Dose Report[17]. This helps us gain an understanding about the Data Elements a typical X-Ray Radiation Dose Report, or another SR, has.

4.1.2 Creating and writing triples to disk

We use the Turtle format to write triples to disk. To recall, a Turtle file is a plaintext file to store RDF triples and provides a few syntactic features that make it pleasant to use. A nice property of Turtle is that its syntax is similar to a SPARQL query.

```
@prefix d2r: <http://dicom2rdf.uniklinik.freiburg.de/rdf/>

d2r:dose-report.dcm d2r:00080016 "1.2.840.10008.5.1.4.1.1.88.67" .
d2r:dose-report.dcm d2r:00181030 "1204_Polytrauma_LAE" .
```

Listing 7: A Turtle file with two triples as they would appear when converting a DICOM file

We introduce the `turtle` crate, a thin abstraction layer to create triples in memory. The `IRI` enum is one of our main building block for constructing triples. It either holds a full IRI, such as

`<http://dicom2rdf.uniklinik-freiburg.de/rdf/dose-report.dcm>`, or a prefixed IRI with a prefix and local component, such as `d2r:dose-report.dcm`.

```rust
pub enum IRI {
    Full(String),
    Prefixed { prefix: String, local: String },
}
```

Listing 8: IRI representation in Rust

Since the object in RDF can be either an IRI or a literal, we provide an additional enum `TripleObject`. A literal comes in two forms: either a plain literal such as `42.0` or `"this"`, or a typed literal such as `"2026-01-19T02:59:44.453Z"^^xsd:dateTime`.

```rust
pub enum PlainLiteral {
    String(String),
    Integer(i64),
    Float(f64),
}
```

```rust
pub struct TypedLiteral {
    lexical: String,
    datatype: IRI,
}
```

```rust
pub enum TripleObject {
    PlainLiteral(PlainLiteral),
    TypedLiteral(TypedLiteral),
    IRI(IRI),
}
```

Listing 9: A triple object, which can either be an IRI or a literal

These building blocks come together with the `Triple` struct, which has a subject, predicate, and object field with appropriate types.

```rust
pub struct Triple {
    pub subject: IRI,
    pub predicate: IRI,
    pub object: TripleObject,
}
```

Listing 10: An RDF triple in Turtle

To recreate the first triple in Listing 7, one would therefore construct the subject, predicate, object, and then construct a triple out of them.

```rust
let s = IRI::Prefixed { prefix: "d2r".into(), local: "dose-report.dcm".into(), };
let p = IRI::Prefixed { prefix: "d2r".into(), local: "00080016".into(), };
let o = TripleObject::PlainLiteral(PlainLiteral::String("1.2.840.10008.5.1.4.1.1.88.67".into()));
let my_triple = turtle::Triple { subject: s, predicate: p, object: o, };
println!("{}", my_triple);
```

Listing 11: Building a Turtle triple from smaller parts

Note that in the source code, one usually passes references to avoid expensive copy operations. However, this would have complicated the types in this section.

Also noteworthy is the last line in Listing 11. Every type that is involved in the `Triple` creation implements the `Display` trait, ensuring that each building block is responsible for a valid Turtle representation of itself, which culminates in a valid triple. For literals, one must escape quotes, backslashes and newlines, as they are not allowed to occur in a literal within a triple.

```
writeln!(&mut buffer, "{}", my_triple)?;
```

Listing 12: Writing a triple to a buffer

Finally, writing a triple to a buffer is simply a matter of calling the `writeln!` macro. This can be `io::stdout`, a `BufWriter`, or anything that implements `Write`. The next chapter will go into more detail.

4.1.3 DICOM file traversal

We can open a DICOM file with dicom-rs and retrieve specific Data Elements as seen in Listing 6. However, we need a systematic way of traversing the DICOM file and writing triples along the way. The `FileDicomObject<T>` implements the `Iterator` trait, which allows us to iterate over all Data Elements. Each `DataElement<T>` holds the corresponding value of a Data Element, but also additional information such as its Tag and VR, which is exactly what we need to create a triple.

```
use crate::turtle::{create_blank_node, iri, triple}

fn process_dicom<W>(writer: &mut W, subject: &IRI, dicom_object: &FileDicomObject<...>)
where W: Write {
  for data_element in dicom_object.iter() {
    // Yields a predicate such as d2r:00080050
    let predicate = iri::new("d2r", data_element.header().tag());
    match data_element.vr() {
      VR::AS => // ...write triple with age-string-specific processing...
      VR::DA => // ...write triple with date-specific processing...
      // ...
      VR::SQ => {
        for (i, item) in data_element.items.iter().enumerate() {
          // Mediator entity for each Item
          let item_mediator = create_blank_node();
          writeln!(writer, "{}", triple(subject, predicate, item_mediator))

          // Each Item has its own Data Elements, which also need a mediator
          let data_elements_mediator = create_blank_node();
          writeln!(writer, "{}", triple(item_mediator, iri::item, data_elements_mediator))

          // Enrich Item mediator with Item's index in the Sequence
          writeln!(writer, "{}", triple(item_mediator, iri::index, i))

          // Repeat the processing step for each Item, where the Item's Data Element
          // mediator is the new parent.
          process_dicom(writer, data_elements_mediator, item)
        }
      }
    }
  }
}
```

Listing 13: A birds-eye view of the recursive DICOM file conversion process

Let us break down what is happening in Listing 13: The processing of a DICOM file always entails passing in something that implements `Write`, a reference to a `parent` IRI, and a reference to a DICOM object. `Write` gives us the flexibility to not only write directly to disk, but buffer the write operations, or wrap the writer around a Gzip encoder to make the resulting files smaller. The `subject` gives us the first component of the triples we build along the way.

We iterate over each Data Element—at this point, we can already create the predicate from the Tag. For each Data Element, there are two outcomes: either it is a primitive value, i.e. not a Sequence, or a Sequence.

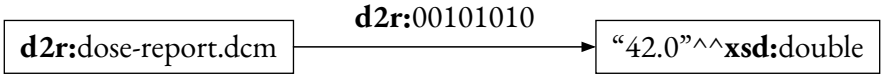

Figure 11: Triple denoting "Patient's Age" from the parent (here: root) to a primitive value

For primitive values, we retrieve the Data Element's value and perform VR-specific handling such as converting an Age String "042Y" to an `f64`, which lets us create the object. With a `subject`, `predicate`, and `object` at hand, we write the triple to the `writer`. Figure 11 shows the resulting triple.

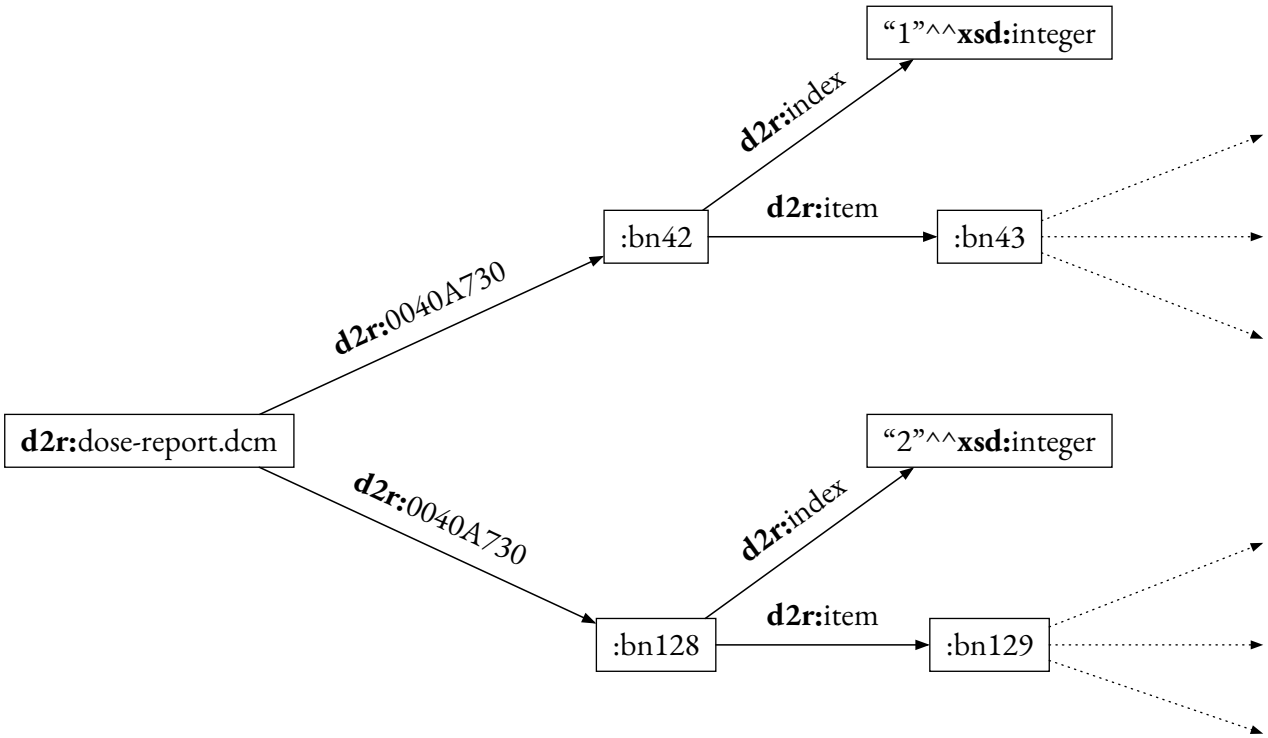

Figure 12: Triples from the parent (here: root) to multiple Sequence Items, each with their own mediator

For Sequence values, we create a blank node for each Item that serves as the mediator between parent and Item. This "Parent–Item" mediator is enriched with the Item's original position in the Sequence via the **d2r:**index predicate to prevent loss of information during conversion. Finally, each "Parent–Item" mediator links to an "Item–Data Elements" mediator via the **d2r:**item predicate. Each of these "Item–Data Elements" mediator serve as the new parent for recursive calls to `process_dicom(...)`. The result is a tree that branches out at Sequences, and terminates at leaves, i.e. primitive values. The resulting triples of processing a Sequence is illustrated in Figure 12.

The `process_dicom` function therefore implements a plain depth-first-traversal of a tree structure. The runtime is O(n), where n is the number of processed Data Elements. It is worth noting that while DICOM SR content is structurally encoded as a tree, individual content items may reference

items other than their parents or children, forming a directed acyclic graph [18, p. 33]. However, this interpretation is out of the scope of this thesis.

4.2 *construct*: Transforming RDF for better queries

We have successfully arrived a checkpoint that gives us some RDF representation of DICOM. This is great, since we can now start a QLever instance and run SPARQL queries against the *convert* dataset to make informed decisions about our next steps.

This section motivates the reasons for why we want to transform the output of *convert*. We provide two queries: The first query highlights issues with triples at the top level. The second query highlights issues with traversing the Content Sequence, where actual measurements are stored. We propose a solution that addresses both issues with top level triples as well as the Content Sequence. Finally, we follow up with an outline on how *construct* implements the proposed solution. The output of *construct* yields an improved version of the query, successfully addressing both issues.

4.2.1 Why *construct*: disjoint collection of top-level triples

Consider Listing 14, a SPARQL query that retrieves a single SR with the study date, study time, patient's age, family name, and given name. This query runs against a dataset that *convert* has produced.

```
PREFIX dicom2rdf: <http://dicom2rdf.uniklinik-freiburg.de/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT * WHERE {
  ?sr dicom2rdf:00080020 ?study_date .
  ?sr dicom2rdf:00080030 ?study_time .
  ?sr dicom2rdf:00101010 ?age .
  ?sr dicom2rdf:person_name ?person_name .
  ?person_name dicom2rdf:pn_family ?pn_family .
  ?person_name dicom2rdf:pn_given ?pn_given .
}
LIMIT 1
```

```
   ?sr           ?study_date    ?study_time    ?age  ?person_name  ?pn_family  ?pn_given
1  bn1160878263  2020-12-19     12:34:56       42.0  bn1160878280  Doe         Jane
```

Listing 14: SPARQL query that retrieves a single SR with the study date, study time, patient's age, family name, and given name.

There are several issues with this query: the SR is a blank node rather than an IRI, i.e. it is not supposed to be an addressable entity, the study date and study time are separate entities even though they belong together semantically. The patient's age is at the top level, while the patient's name has its own mediator entity. The predicates use our own namespace that mirrors DICOM tags, even though there are established ontologies where the meaning of some predicates is much clearer.

For example, schema.org[19] provides many predicates for concepts related to a person, such as **schema:**familyName. In general, it is a rather disjoint collection of triples.

However, we have undeniably reached a stage where there is no clear-cut right approach and personal preferences play a big role in the design of a schema. Suppose we introduce a mediator node that represents a patient, and that patient node links to a family name via **schema:**familyName. One could, for example, argue that the age from the SR does not belong to the mediator node, as the number is a momentary snapshot of the patient's age at the time of examination. DICOM also has a "Patient's Birth Date" attribute, and having both the patient's age at the time of examination as well as birth date via the same mediator node may be counterintuitive.

Nevertheless, we recognize that there are at least *some* triples that need restructuring. We address this issue with a few simple operations, such as joining pairs of **xsd:**gDate and **xsd:**gTime objects to **xsd:**dateTime, creating an IRI for the SR from values that are known to be globally unique such as, SOP Instance UID, normalizing demographic data such as patient weight and height to kg and cm using heuristic cut-off values, and more. It is important to note that the decisions in this context are highly subjective.

4.2.2 Why *construct*: too many required hops to reach concepts and measurements

A much more important reason for why we need to construct new triples is the retrieval of measurements from the Content Sequence. Consider Listing 15, a convoluted SPARQL query that retrieves the numeric value of all Mean $CTDI_{vol}$ measurements and runs against a dataset that *convert* has produced. The point is not to understand how the query works, but to show the amount of triple patterns needed to extract a single measurement.

The triple patterns reveal the DICOM file's inner structure: Each triple represents the relationship between a Data Set and a Data Element's value, where the subject is the Data Set, the predicate is the Data Element, and the object is the value contained in the Data Element. As such, accessing values in a DICOM files is equivalent to using a DICOM library such as pydicom or dicom-rs.

```
PREFIX dcm: <https://dicom.nema.org/resources/ontology/DCM/>
PREFIX dicom2rdf: <http://dicom2rdf.uniklinik-freiburg.de/>
SELECT ?mean_ctdivol_numeric_value WHERE {
    ?dicom_sr dicom2rdf:0040A730 ?root_container .
    ?root_container dicom2rdf:item ?ct_acquisition .
    ?ct_acquisition dicom2rdf:0040A043 ?ct_acquisition_concept_name .
    ?ct_acquisition_concept_name dicom2rdf:00080102 dcm: .
    ?ct_acquisition_concept_name dicom2rdf:00080100 "113819" .
    ?ct_acquisition dicom2rdf:0040A730 ?container2 .
    ?container2 dicom2rdf:item ?ct_dose .
    ?ct_dose dicom2rdf:0040A043 ?ct_dose_concept_name .
    ?ct_dose_concept_name dicom2rdf:00080102 dcm: .
    ?ct_dose_concept_name dicom2rdf:00080100 "113829" .
    ?ct_dose dicom2rdf:0040A730 ?ct_dose_container .
    ?ct_dose_container dicom2rdf:item ?mean_ctdivol .
    ?mean_ctdivol dicom2rdf:0040A043 ?mean_ctdivol_concept_name .
    ?mean_ctdivol_concept_name dicom2rdf:00080102 dcm: .
    ?mean_ctdivol_concept_name dicom2rdf:00080100 "113830" .
    ?mean_ctdivol dicom2rdf:0040A300 ?mean_ctdivol_measured_value .
    ?mean_ctdivol_measured_value dicom2rdf:004008EA ?mean_ctdivol_measurement_units .
    ?mean_ctdivol_measured_value dicom2rdf:0040A30A ?mean_ctdivol_numeric_value .
}
```

Listing 15: SPARQL query that retrieves all Mean $CTDI_{vol}$ measurements on the dataset produced by *convert*.

This query has two main issues. Beginning from the DICOM object's root, the query needs **17 triple patterns** to reach the measurement. Additionally, the query's triple patterns do not reflect the semantics imposed by the RDSR template, as shown in Figure 10. In order to write a query such as Listing 15, one has to be intricately familiar with the meaning behind codes such as "113819" and how Data Elements are interconnected. The abstraction of DICOM's file format leaks through the schema.

Despite needing 17 triple patterns, Listing 15 does not yet yield useful results. The Mean $CTDI_{vol}$ measurements are not comparable across RDSRs without stratifying the patient collective or, at the very least, the Protocol Name. A useful SPARQL query that runs against the *convert* dataset easily needs over a hundred triple patters.

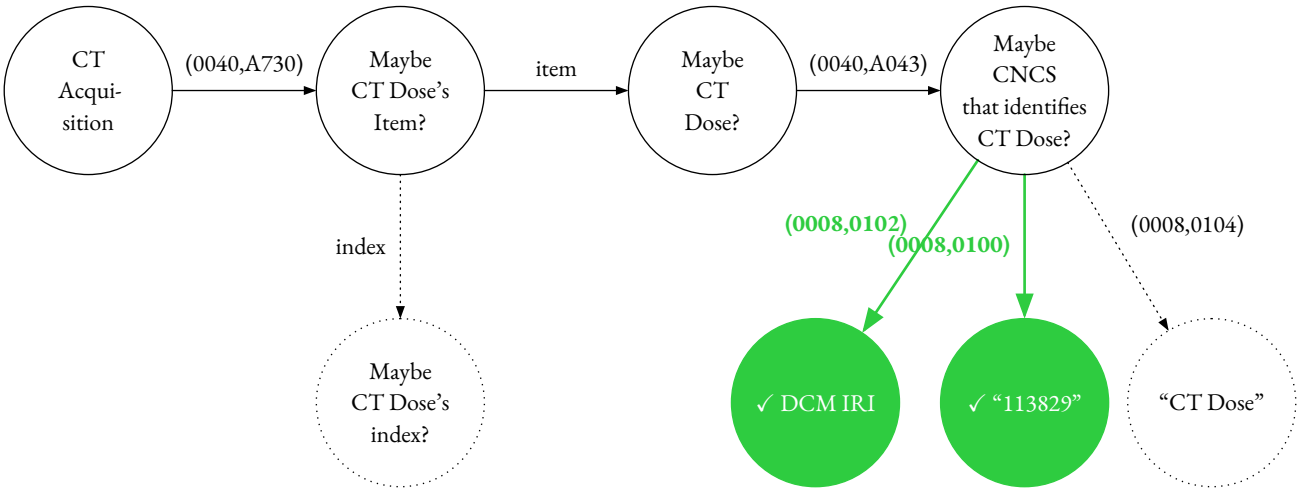

Figure 13: Five hops are required to ensure that the destination concept is a CT dose. The index and Code Meaning (0008,0104) is not strictly required.
(CNCS = Concept Name Code Sequence)

Upon closer inspection, we see recurring patterns. Consider Figure 13, which illustrates the triple patterns of Listing 15 that lead from *?ct_acquisition* to *?ct_dose*. In order to go from a starting concept to a destination concept, one has to write triple patterns that connect the starting concept to its Content Sequence (0040,A730), and from its Content Sequence to its Item. Then, one has to make sure that the Item is indeed the destination concept by looking up its Concept Name Code Sequence (0040,A043), followed by looking up its Coding Scheme (0008,0102) and its Concept Code (0008,0100). The important part is recognizing that there is no information gain up until the point where the values of Coding Scheme and Concept Code are known.

We address this issue in two ways: First, each hop gains as much information as possible. Second, hops that do occur are appropriately labeled to leverage QLever UI's explorable autocomplete feature.

Our approach works as follows: we concatenate **dcm:** (0008,0102) and "113829" (0008,0100) to form the predicate **dcm:**113829. Next, we enrich this predicate with the label "CT Dose" that (0008,0104) provides. After these transformations, we can remove the Concept Name Code Sequence without loss of information. Figure 14 shows the result of this transformation.

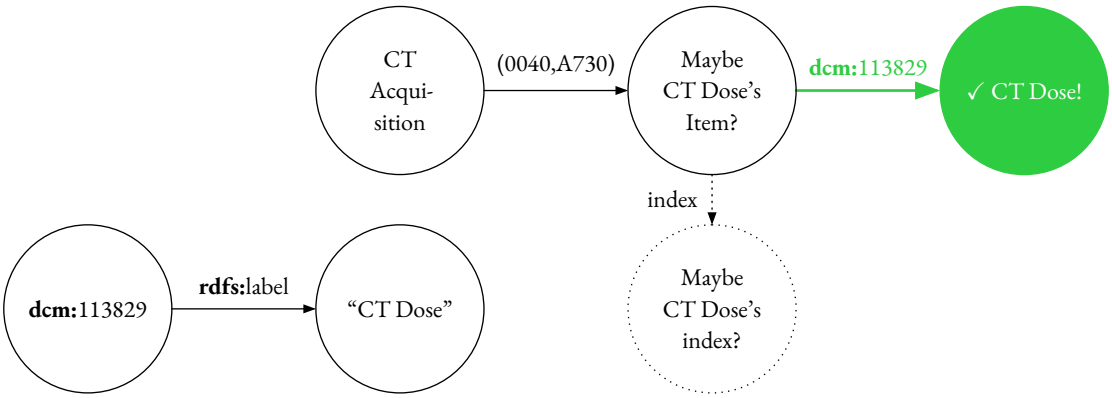

Figure 14: CNCS has been eliminated and its information encoded elsewhere

However, we can perform one more transformation.

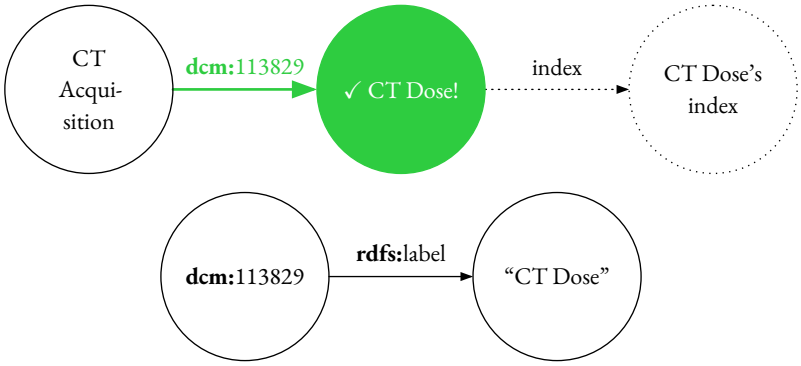

Figure 15: Intermediate node has been contracted and the index information moved to the concept, if needed

The result is a set of triples that only requires a single hop to go from one concept to the next, as seen in Figure 15.

4.2.3 Using SPARQL CONSTRUCT queries to form new triples

In Section 2, we have already presented SPARQL CONSTRUCT queries as a tool to transform a knowledge graph and form new triples that is a better fit for some world model. In this section, we present our approach both for the top-level triples, as well as Content Sequence triples.

We introduce a small abstraction for creating SPARQL CONSTRUCT queries.

```
let base = ConstructQuery::new()
    .with_prefixes(prefixes(config))
    .with_where(vec![
        "?level0 a dicom2rdf:DocumentRoot .",
        "?level0 dicom2rdf:00080018 ?sopInstanceUID .",
```

```
        r#"BIND(IRI(CONCAT(STR(rad:), "sopInstance/", ?sopInstanceUID)) AS ?level0IRI)"#,
    ]);
```

By creating a base query, we can extend onto it and dynamically create new queries.

```
base.clone()
    .with_construct(vec![
        "?level0IRI sct:71388002 ?procedure .",
        "?procedure rdfs:label ?procedureCodeMeaning .",
    ])
    .with_where(vec![
        "?level0 dicom2rdf:00081032/dicom2rdf:00080102 ?procedureCodingScheme .",
        "?level0 dicom2rdf:00081032/dicom2rdf:00080100 ?procedureCode .",
        "?level0 dicom2rdf:00081032/dicom2rdf:00080104 ?procedureCodeMeaning .",
        "BIND(IRI(CONCAT(STR(?procedureCodingScheme), ENCODE_FOR_URI(?procedureCode)))
AS ?procedure)",
    ])
```

This allows us to programmatically create CONSTRUCT queries that reach any container nesting within a DICOM file.

Let us now look at the same query, but on the newly constructed dataset.

```
PREFIX dcm: <https://dicom.nema.org/resources/ontology/DCM/>
PREFIX dicom2rdf: <http://dicom2rdf.uniklinik-freiburg.de/>
SELECT ?mean_ctdivol_numeric_value WHERE {
    ?dicom_sr dcm:113819 ?ct_acquisition .
    ?ct_acquisition dcm:113829 ?ct_dose .
    ?ct_dose dcm:113830 ?mean_ctdivol .
    ?mean_ctdivol qudt:unit ucum:mGy .
    ?mean_ctdivol qudt:numericValue ?mean_ctdivol_numericvalue .
}
```

Listing 16: SPARQL query that retrieves all Mean $CTDI_{vol}$ measurements on the new dataset produced by *construct*.

Listing 17 provides a semantically identical version of Listing 16, but written by someone who is familiar with the syntactic sugar that Turtle provides.

```
PREFIX dcm: <https://dicom.nema.org/resources/ontology/DCM/>
PREFIX dicom2rdf: <http://dicom2rdf.uniklinik-freiburg.de/>
SELECT ?mean_ctdivol WHERE {
    # Omit intermediate CT Acquisition object and subject.
    ?dicom_sr dcm:113819/dcm:113829 ?ct_dose .
    # Omit intermediate CTDIvol subject, directly match its predicate-objects.
    ?ct_dose dcm:113830 [ qudt:unit ucum:mGy ; qudt:numericValue ?mean_ctdivol ] .
}
```

Listing 17: Same as Listing 16 but with Turtle's syntactic sugar

1. It needs five triple patterns, matching its natural language counterpart.
2. On a small dataset with 10000 DICOM files, it already runs **over 5 times faster**.

4.3 Optimizations: Scaling from Thousands to Millions of DICOM SRs

Various optimizations have been performed to cope with the number of files we intend to process. This section presents some of the optimizations. Their effect will be apparent in Section 5.

4.3.1 Parallelize conversion

In its naïve implementation, *convert* processes one file after another. The processing of one DICOM file does not depend on the processing result of another file. Therefore, the procedure that *convert* implements is embarrassingly parallel. We use `rayon`, a data parallelism library for Rust. It uses a work stealing strategy to let other worker threads steal work from each other. This is especially important for DICOM files, where their size and therefore processing duration can vary drastically.

In order to feed the worker threads with file paths as fast as possible, we use `walkdir` and stream the paths to the processing pipeline as they come. One downside is that we cannot estimate how long the conversion will take as the total number of files is unknown in advance.

4.3.2 Spending CPU time to cope with I/O boundedness

When scaling from one thread to dozens, we noticed that the disks could not keep up. We resolved this issue by simply adding a Gzip compression step to the file writing process and trading CPU time for less disk contention.

4.3.3 Progress Logging with multiple workers

An interesting effect when multiple worker threads are involved is knowing the overall progress of a task. As more worker threads were involved, we noticed that an atomic counter that is incremented becomes a severe bottleneck.

To solve this issue, we use a multiple-producer-single-consumer (mpsc) channel in Rust. Each worker receives a channel to send a progress report to. When a certain milestone is met—determined by a fixed

number, divided by the number of available worker threads—each worker sends a progress report to a centralized worker. This way, lock contention is avoided and progress is only reported in large intervals.

4.3.4 Starting multiple smaller QLever instances

We have noticed that our *construct* stage consumes a lot of memory for large QLever instances, exceeding even the memory of a 128 GB server. We decided to design our pipeline such that smaller QLever instances are started. This way, a CONSTRUCT query requires less memory.

4.4 Orchestrating the *dicom2rdf* pipeline

With *convert*, *construct*, and performance optimizations in place, we introduce the ***dicom2rdf*** pipeline.

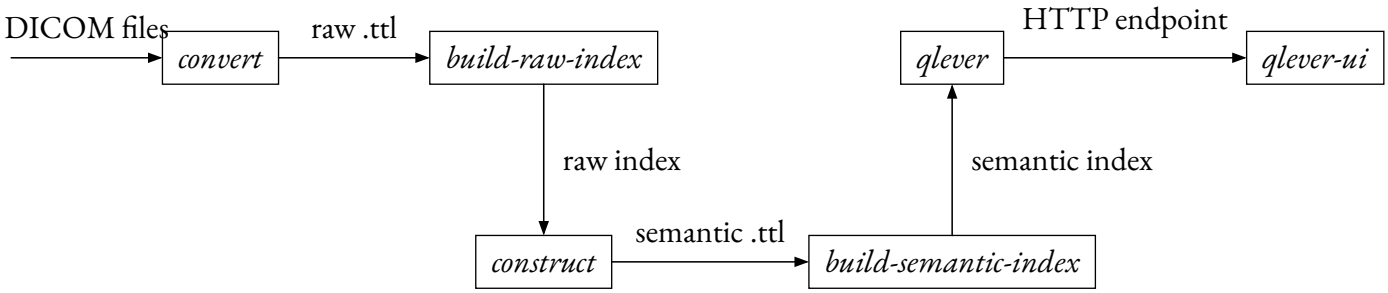

Figure 16: *dicom2rdf* pipeline

Various dependency directives and health checks ensure that only a single command is needed to go from a collection of DICOM files to a queryable interface: `docker compose up --build`.

CHAPTER 5

Evaluation

This chapter evaluates the performance of the dicom2rdf pipeline and its practicability in a clinical context. In Section 5.1, we observe the baseline performance of the pipeline as a whole and its parts in isolation. We compare the ***convert*** *part of our pipeline to existing approaches and observe its performance characteristics under controlled change. In Section 5.2, we put the capabilities of the knowledge graph in a clinical context to the test. We first define a set of tasks related to dose management and outcome-based , provide a reasoning for* why *these tasks are interesting, and assess whether or not these .*

5.1 Performance Evaluation

5.1.1 Setup

The evaluation is performed on an **Ubuntu 24.04** server with an **AMD Ryzen 9 5900X 12-Core Processor CPU** and **128 GB memory**. We evaluate performance on two datasets, one for comparison purposes with dcm2rdf, and one for a real-world end-to-end test of deploying dicom2rdf.

The first dataset was obtained from the National Cancer Institute (NCI). Due to licensing constraints, the dataset cannot be redistributed and must be obtained from the NCI Imaging Data Commons[3]. The specifics of the NCI dataset is given in Table 4. We note that the NCI dataset does not exclusively contain Structured Reports, but predominantly CT modalities. However, we argue that in the context of performance measurements, this does not make a difference for the comparison as both dicom2rdf's *convert* and dcm2rdf, the program we compare against, do not process pixel data. As public DICOM files are difficult to come by, these are the circumstances we have to live with. A breakdown of modalities is given in Table 5. The directories containing the files are placed in `/local/data-ssd/aydina/nci/01,02,...,10}`, where each directory contains the full collection as many times as the directory states. For example, `01` contains the collections `LUNG1-001`, `LUNG1-002`, ..., `LUNG1-010`. `02` contains the collections `LUNG1-001`, `LUNG1-002`, ..., `LUNG1-010`, and repeats a second time with a file name suffix. This is to ensure that the programs have enough data to work work through.

[3] https://portal.imaging.datacommons.cancer.gov/explore/

Collection Name	Case ID	No. Series	No. Files	File Size
NSCLC Radiomics	LUNG1-001	11	144	111.50 MB
	LUNG1-002	11	122	103.70 MB
	LUNG1-003	11	118	99.41 MB
	LUNG1-004	11	125	102.16 MB
	LUNG1-005	11	102	87.70 MB
	LUNG1-006	11	125	107.15 MB
	LUNG1-007	11	140	115.41 MB
	LUNG1-008	11	125	103.20 MB
	LUNG1-009	11	116	95.46 MB
	LUNG1-010	11	102	87.36 MB

Table 4: Test dataset obtained from NCI

Case ID	CT	SR	SEG	RTSTRUCT
LUNG1-001	134	5	4	1
LUNG1-002	111	6	4	1
LUNG1-003	107	6	4	1
LUNG1-004	114	6	4	1
LUNG1-005	91	6	4	1
LUNG1-006	114	6	4	1
LUNG1-007	129	6	4	1
LUNG1-008	114	6	4	1
LUNG1-009	105	6	4	1
LUNG1-010	91	6	4	1

Table 5: Modality breakdown

The second dataset was provided by the University Medical Center Freiburg and predominantly contains Structured Reports, with specifics given in Listing 18. The directories containing the files are placed evenly distributed in `/local/data-ssd/aydina/dicom-ukf/{00,01,...,09,0a,0b,...,ff}`.

```
PREFIX dicom2rdf: <http://dicom2rdf.uniklinik-freiburg.de/>
SELECT ?modality (COUNT(?modality) AS ?count_modality) WHERE {
  ?dicom_file dicom2rdf:00080060 ?modality .
}
GROUP BY ?modality
ORDER BY DESC(?count_modality)
```

Modality	Count
SR	7,139,423
CT	29,551
PR	251
CR	148
MR	14
XA	11
DX	3
DR	2
OT	2
RTDOSE	2
PT	1
PT/CT	1

Listing 18: Dataset provided by the University Medical Center Freiburg, numbers obtained with SPARQL query

5.1.2 Methodology

In the first part of the performance evaluation, we compare *convert* against *dcm2rdf*, as they both perform the task of converting DICOM files to RDF with a similar schema—that is, exposing the hierarchical structure as RDF triples. We first process `LUNG1-001`, then `LUNG1-001 + LUNG1-002`, and so on, until we process all 10 collections.

In the second part, we benchmark convert with varying configurations.

Each benchmark is always run 12 times, the first two results are discarded, and the average is taken.

5.1.3 Benchmark: *convert* vs. *dcm2rdf*

The *dicom2rdf* repository provides a `benchmark-convert` image and a benchmark script. We initiate the benchmark with the command given in Listing 19. The version we use has the commit hash `f6342bc1188f3ad35ce89cd7e5e96f894a7418b3`.

```
mkdir -p data/benchmark-output
docker build --target benchmark-convert -t dicom2rdf-bench .
docker run --rm \
    -v /local/data-ssd/aydina/nci/01:/input:ro \
    -v ./data/benchmark-output:/output dicom2rdf-bench
```

Listing 19: Initiating the dicom2rdf benchmark for LUNG1-001

The benchmark results are written to a CSV file in `./data/benchmark-output/report.csv` with various configurations. In this benchmark, we consider the result for 24 threads.

5.1.4 Preparation: *dcm2rdf*

We have cloned the *dcm2rdf* repository to a local directory. The version we use has the commit hash `78f7f4e177ef313488c3d3650435874b7c73bf18`. We provide a Dockerfile for the purposes of benchmarking *dcm2rdf*.

```
FROM maven:3.9-eclipse-temurin-21 AS builder

WORKDIR /app
COPY pom.xml .
COPY src ./src

RUN mvn -Pjar clean package -q -DskipTests

FROM eclipse-temurin:21-jre

WORKDIR /app
COPY --from=builder /app/target/dcm2rdf-1.2.0.jar /app/dcm2rdf.jar

ENTRYPOINT ["java", "-jar", "/app/dcm2rdf.jar"]
```

Listing 20: Dockerfile used to benchmark *dcm2rdf*

```
docker build -t dcm2rdf .
docker run --rm \
    -v /local/data-ssd/aydina/nci/01:/input:ro \
    -v /local/data-ssd/aydina/output-dcm2rdf/:/output \
    dcm2rdf -src /input -dest /output -t 24 -status
```

Listing 21: Initiating the dcm2rdf benchmark for LUNG1-001

The conversion runtime is written to `stdout`.

Num files	Runtime dcm2rdf (ms)	Runtime convert
1219	9123	27

Table 6: Performance comparison dcm2rdf vs. convert

5.2 *dicom2rdf* in a clinical context

This section evaluates the practical aspects of *dicom2rdf* as a tool to answer clinically relevant questions.

5.2.1 Dose Management

One of the main motivations for this thesis has been the prospect of retrieving dose measurements on a per-acquisition-protocol level. We have already discussed that while filtering by protocol name is the norm, it includes non-diagnostic doses that distort the actual diagnostic dose values. The result is a web application with various predefined reports, as shown in Figure 17.

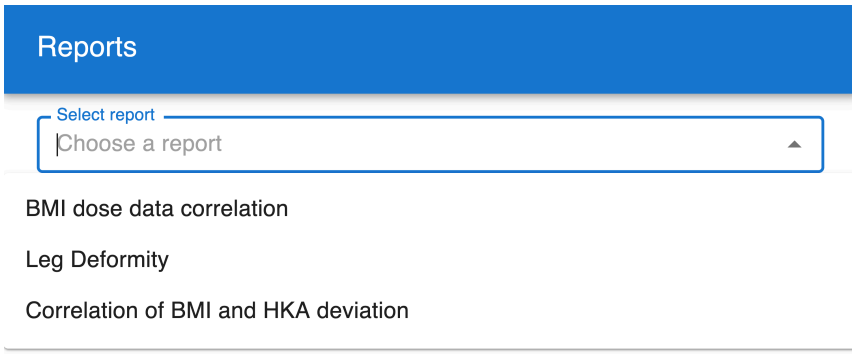

Figure 17: A selection of reports

One such report is the “BMI dose data correlation”, as shown in Figure 18. The idea is to plot both $CTDI_{vol}$ and SSDE dose values for a specific acquisition protocol against BMI values of patients. The plot shows two intended phenomena:

1. The emitted radiation dose correlates with increasing BMI. A higher BMI implies a larger patient, which attenuates more X-rays. CT scanners use so-called automatic exposure control to modulate tube current in real-time. If higher attenuation is detected, tube current is increased.
2. The size-specific dose estimate (SSDE) is derived by $CTDI_{vol}$ multiplied by a conversion factor, which depends on the patient’s detected diameter. While $CTDI_{vol}$ measure the CT scanner’s output using a standardized output, SSDE aims to correct for anatomical differences, with the goal of providing a more accurate estimate than $CTDI_{vol}$. Therefore, SSDE corrects the underestimated value for smaller patients and vice versa.

Figure 18: BMI-dose data correlation

In addition, we added another option to filter by device. This allows a medical physics expert to assess how differently devices perform and take appropriate action.

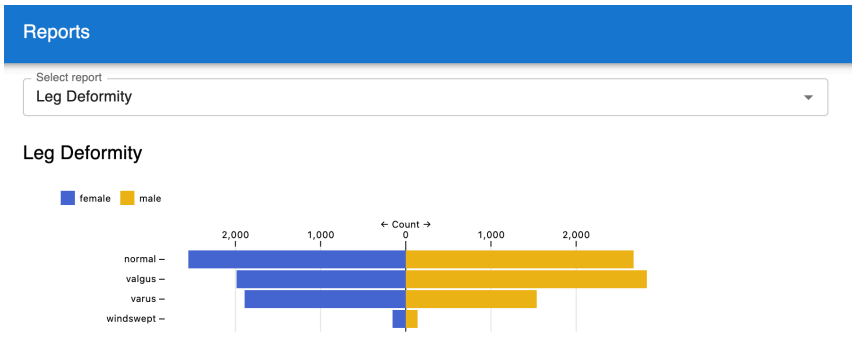

Figure 19: Leg deformity

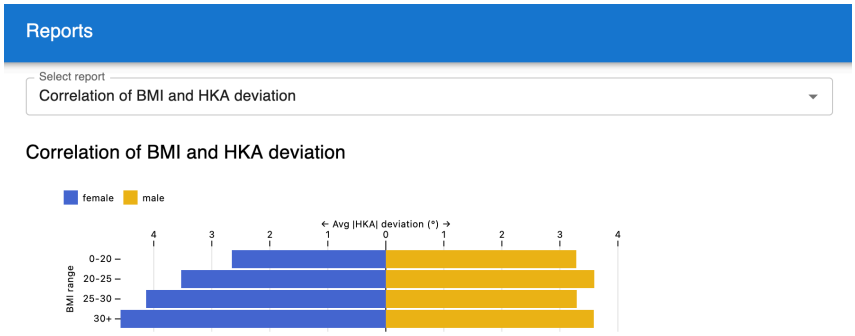

Figure 20: BMI-hip-knee angle (HKA) deviation

The remaining two reports derive data from leg angle measurements. The corresponding Structured Reports do not adhere to any template such as TID 1500, but can still be queried due to the flexibility of RDF and performance of QLever.

CHAPTER 6

Conclusion and Future Work

This chapter concludes the thesis and outlines future work on dicom2rdf. Section 6.1 summarizes our contribution and highlights the potential use-cases of dicom2rdf in diagnostic reporting. Section 6.2 lists future work that needs to be completed in order of priority to improve dicom2rdf even further.

6.1 Conclusion

In retrospect, dicom2rdf has proven itself to be a useful tool to retrieve data from DICOM Structured Reports. The core strength has been the fast execution on queries on very large datasets: the intermediate RDF of *convert* produced over 7 billion triples and was, in terms of querying speeds, still useful enough to inform our decision-making for the design of the simplified RDF with through *construct*. The transformations of *construct* has pushed the triple count down to a billion triples by focusing on maximizing information gain between hops, e.g. from *CT Acquisition* to *CT Dose*, where actual irradiation measurements are stored. At the same time, the transformation improved the ergonomics of writing a query by labeling each predicate. Paired with QLever UI, this made DICOM Structured Reports a breeze to explore and traverse.

6.2 Future Work

However, there is a lot of work that can be done to improve dicom2rdf even further. The following is a collection of work items that emerged over the time of creating dicom2rdf. Some of the work items relate to technical aspects, while other work items relate to usefulness in a real-world setting.

6.2.1 Rethink CONSTRUCT strategy

The *construct* part of our pipeline produces useful triples, but has two severe downsides that affect both performance and practical utility. As the nesting level of each CONSTRUCT query increases, the same paths are traversed repeatedly. This additional cost increases with each nesting level. This increase in cost was the reason we had to split the *construct* stage to multiple QLever instances—to make the execution of a query on 7 billion triples even feasible without running into memory issues.

We propose the following idea to address the issue.
1. Run the CONSTRUCT queries for the first nesting level, as we do already
2. Add the Sequence Item that must be traversed together with its precomputed IRI to some `frontier`
3. Repeat step 1, but starting from all nodes in `frontier`. Use the precomputed IRI to compute a new level IRI for the next Sequence Items.

This yields a series of CONSTRUCT queries that are computationally manageable even on a single QLever instance with 7 billion triples.

Regarding step 2: In Section 4.1.3, we described how we create blank nodes that serve as the mediator between parent and Item. This was not completely true: we create IRIs in the form `<bn42>`. These look like blank nodes, but are regular full IRIs, which makes them addressable. The reason for this is that blank node labels are scoped to a result set. Creating such IRIs was a preparatory technical decision for this working item.

We estimate that a working day is needed to incorporte this new strategy to dicom2rdf. Reworking the *construct* part of the pipeline is the single most important work item as it sets up other work item.

6.2.2 Take containers within non-container Value Types into account

The CONSTRUCT queries we generate make the general assumption that measurement values such as numbers or text cannot have containers themselves. However, this is not true, as evident by the size-specific dose measurement: it is a single measure, but it is *derived from* other measures. The measures it is derived from are within a container of that number.

The reason for this assumption was two-fold: First, the derived measure such as SSDE was simply the most interesting one and sufficient for the first version of dicom2rdf. Second, it was computationally not feasible, as the CONSTRUCT queries would've had to express that they can not only descend further via containers, but also non-containers. This would've blown up the search space.

With the changes in Section 6.2.1, capturing these derived measures within non-containers becomes feasible again. Once Section 6.2.1 is implemented, we estimate that this work item takes 2–3 hours to complete.

6.2.3 Capture values of remaining Value Types

In this version of dicom2rdf, we focused on five major Value Types: TEXT, NUM, CODE, UIDREF, and CONTAINER. However, there are 11 more Value Types and capturing all of them is a natural next step. Again, this work item depends on Section 6.2.1 and we estimate that a working day is needed for completion. The required time is not due to technical challenges, but thinking about how to best represent certain values in idiomatic RDF, such as modeling SCOORD using the Well-Known Text (WKT) representation.

6.2.4 Auto-tune parameters such as compression level

We have seen that compression can help with increasing the throughput of *convert*, but finding the correct value is a manual process. One idea is to test throughput with various compression ratios in an isolated setting in a separate stage as part of the pipeline. We estimate that building such a stage takes 3–4 hours.

There are many more and small work items, but the three listed work items above are the major ones.

Acknowledgments

I want to thank Prof. Dr. Hannah Bast for her continuous guidance, which helped me develop an understanding of the subject and enabled me to work on this topic. Additionally, I want to thank Dr. Philipp Arnold, whose medical expertise gave me an appreciation for the field of medical imaging. Finally, I want to thank my family for their support throughout my studies.

Bibliography

[1] Bundesamt für Strahlenschutz, "What does the ALARA principle mean?." Accessed: Jan. 17, 2026. [Online]. Available: https://www.bfs.de/SharedDocs/FAQs/BfS/EN/ion/ion/alara.html

[2] World Health Organization, "Ionizing radiation and health effects." Accessed: Jan. 17, 2026. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/ionizing-radiation-and-health-effects

[3] R. Smith-Bindman *et al.*, "Projected Lifetime Cancer Risks From Current Computed Tomography Imaging," *JAMA Internal Medicine*, vol. 185, no. 6, pp. 710–719, 2025, doi: 10.1001/jamainternmed.2025.0505.

[4] "Council Directive 2013/59/Euratom of 5 December 2013 laying down basic safety standards for protection against the dangers arising from exposure to ionising radiation." [Online]. Available: https://eur-lex.europa.eu/eli/dir/2013/59/oj

[5] T. D. standard authors, "The DICOM Standard." Accessed: Jan. 18, 2026. [Online]. Available: https://dicomstandard.org/

[6] I.-K. Sebelego, S. Acho, B. van der Merwe, and W. I. D. Rae, "FACTORS INFLUENCING SIZE-SPECIFIC DOSE ESTIMATES OF SELECTED COMPUTED TOMOGRAPHY PROTOCOLS AT TWO CLINICAL PRACTICES IN SOUTH AFRICA," *Radiation Protection Dosimetry*, vol. 199, no. 7, pp. 588–602, 2023, doi: 10.1093/rpd/ncad059.

[7] E. Z. Dalah *et al.*, "Challenges Associated with Effective Implementation of CT Dose Check Standards and Radiation Monitoring Index in Computed Tomography: Healthcare Sector Experience," *Healthcare*, vol. 10, no. 10, 2022, doi: 10.3390/healthcare10101970.

[8] F. Ria *et al.*, "Expanding the Concept of Diagnostic Reference Levels to Noise and Dose Reference Levels in CT," *AJR American Journal of Roentgenology*, vol. 213, no. 4, pp. 889–894, 2019, doi: 10.2214/AJR.18.21030.

[9] N. Saltybaeva and H. Alkadhi, "Clinical dose reference levels: developing and application," in *EuroSafe Imaging 2019*, 2019. doi: 10.26044/esi2019/ESI-0061.

[10] J. W. Prescott, "Quantitative Imaging Biomarkers: The Application of Advanced Image Processing and Analysis to Clinical and Preclinical Decision Making," *Journal of Digital Imaging*, vol. 26, no. 1, pp. 97–108, 2013, doi: 10.1007/s10278-012-9465-7.

[11] A. Bejaoui *et al.*, "Metadata Quality in Imaging Trials: A Knowledge Graph-Based Approach," *Studies in Health Technology and Informatics*, vol. 327, pp. 593–594, 2025, doi: 10.3233/SHTI250408.

[12] E. Bremer, "dcm2rdf: DICOM to RDF Converter." Accessed: Jan. 19, 2026. [Online]. Available: https://github.com/ebremer/dcm2rdf

[13] M. Brunnbauer, "dicom2rdf." Accessed: Jan. 19, 2026. [Online]. Available: https://github.com/Bonubase/dicom2rdf

[14] NEMA, "A.35.1 Basic Text SR IOD." 2013. [Online]. Available: https://dicom.nema.org/medical/dicom/current/output/chtml/part03/sect_A.35.html#sect_A.35.1

[15] NEMA, "C.7.1.1 Patient Module." 2013. [Online]. Available: https://dicom.nema.org/medical/dicom/current/output/chtml/part03/sect_C.7.html#sect_C.7.1.1

[16] NEMA, "DICOM PS3.4: Service Class Specifications - Part 4: Service-Class User Definitions, Section B.5: Standard SOP Classes." 2013. [Online]. Available: https://dicom.nema.org/dicom/2013/output/chtml/part04/sect_B.5.html

[17] NEMA, "A.35.8 X-Ray Radiation Dose SR IOD." 2013. [Online]. Available: https://dicom.nema.org/dicom/2013/output/chtml/part03/sect_A.35.html#sect_A.35.8

[18] D. A. Clunie, *DICOM Structured Reporting*. PixelMed Publishing, 2000. [Online]. Available: https://pixelmed.com/srbook.html

[19] Schema.org, "Schema.org." Accessed: Jan. 18, 2026. [Online]. Available: https://schema.org/

Glossary

Term	Definition
ALARA	As Low As Reasonably Achievable - a principle of radiation protection
CT	Computed Tomography - a medical imaging procedure using X-rays
DICOM	Digital Imaging and Communications in Medicine - a standard for handling medical imaging information
DICOM SR	DICOM Structured Report
DMS	Dose Management System
DRL	Diagnostic Reference Level - a dose level used as a reference point for comparison
IOD	Information Object Definition - object-oriented abstract data model, e.g. Patient IOD
MPE	Medical Physics Expert
RDSR	Radiation Dose Structured Report - a DICOM object that records radiation dose information
RDF	Resource Description Framework - a framework for representing information on the web
SPARQL	SPARQL Protocol and RDF Query Language

Full SPARQL Queries

```
PREFIX schema: <https://schema.org/>
PREFIX dicom2rdf: <http://dicom2rdf.uniklinik-freiburg.de/>
PREFIX dcm: <https://dicom.nema.org/resources/ontology/DCM/>
PREFIX qudt: <http://qudt.org/schema/qudt/>
PREFIX rad: <http://uniklinik-freiburg.de/rdf/radiologie/>
PREFIX unk: <https://example.com/unknown/>
PREFIX schema: <http://schema.org/>
SELECT ?series_instance_uid ?mean_ctdivol ?ssde ?bmi ?device_subject_model_name WHERE {
  ?sr dcm:121139 "SR" .
  ?sr dcm:112002 ?series_instance_uid .
  ?sr rad:patient ?patient .
  ?patient rad:age ?age .
  FILTER (?age ⩾ 18)
  ?patient schema:weight ?weight .
  ?patient schema:height ?height .
  BIND (?height / 100 AS ?height_m)
  BIND (?weight / (?height_m * ?height_m) AS ?bmi)
  FILTER (?bmi ⩾ 12 && ?bmi ⩽ 80)
  ?sr dcm:121195 ?device_subject_model_name .
  VALUES ?acquisition_protocol { "Abdomen pv" }
  {
    ?sr unk:99CT_SOMX_2 ?ct_acquisition_report .
    ?ct_acquisition_report dcm:125203 ?acquisition_protocol .
    ?ct_acquisition_report dcm:113819 ?ct_acquisition .
    ?ct_acquisition dcm:113822 ?ct_acquisition_parameters .
    ?ct_acquisition_parameters dcm:113830/qudt:numericValue ?mean_ctdivol .
    ?ct_acquisition dcm:113930/qudt:numericValue ?ssde .
  }
  UNION {
    ?sr dcm:113819 ?ct_acquisition .
    ?ct_acquisition dcm:125203 ?acquisition_protocol .
    ?ct_acquisition dcm:113829 ?ct_dose .
    ?ct_dose dcm:113830/qudt:numericValue ?mean_ctdivol .
    ?ct_dose dcm:113930/qudt:numericValue ?ssde .
  }
}
```