NTNU
Det skapende universitet

Evaluere bruk av brukermodeller ved fulltekstsøk i bibliografiske data

Tri Minh Nguyen

Master i informatikk
Innlevert: januar 2015
Hovedveileder: Trond Aalberg, IDI

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

I would like to thank my supervisor Trond Aalberg for the help and support in realizing this project. I would also like to thank all my friends and family, and all who have been cheering for me, for their continued support.

A special thanks to Hanne Gunby for helping me correcting and dressing up language errors, for her support throughout the study, and for being a good friend!

Abstract

The topic of this thesis is search methods using keyword search on unstructured data. The key idea is to define rules for a specific type of data that will provide a search method with better accuracy. It seeks to construct a subtle data model using bibliographic data.

Other search systems are analyzed and compared with each other. This thesis explores methods and techniques using full-text indexing on next generation *bibliographic information systems*, and seeks to propose a search method with accompanying definitions for a user method.

Problems in the search methods are discussed, and recommended future research is described.

Table of Contents

List of Figures

Listings

Overview and summary of the thesis

Chapter 1: Introduction

The first chapter is an introduction to the topic of the thesis. This chapter will cover the context and background for this study, and introduce the research objectives. *User behaviour*, *keyword search*, and concepts like FRBR and the database and its mechanism are introduced. The main contributions of the thesis are listed in this chapter.

Chapter 2: Theoretical Background

Internet search engines have revolutionized the way people finds information on the web, and in general, on screen. Chapter 2 start out by introducing the data model, and further skim the surface of keyword search and keyword search in graphs. The last section in this chapter shows some examples on how online libraries looks like today, and how they acts when in use.

Chapter 3: The Search Method

Chapter 3 details the proposed *search method*. It will walk through the search method, from its idea, and toward to its functionality. Later on it will discuss some of the search scnearios it will come across.

Chapter 4: Implementation

Chapter 4 describes how the system is implemented, and its technology stack. It will further go into some details about the technologies, and give an overview of the system itself.

Chapter 5: Problems in the *Search Method*

While covering the search method and its technology, problems occured. These are problem that did not get any priorities in this study. Future research on these are proposed in chapter 7.

Chapter 6: Process evaluation

Chapter 6 will discuss the process during this study. Thoughts and plain summary on the main stages of this study is presented as is.

Chapter 7: Conclusions

The last chapter will conclude the thesis by supporting the arguments: the search method, the cusomtized user model. Future research is presented in the very end of this chapter.

1

Introduction

Searching, retrieving, processing are all hot topics in computer science. And discussions on how to retrieve data often arises when having information lying around. This study will focus on retrieving bibliographic information based on a more subtle data model.

The information surounding us are enormous. Questions on how to structure and finding the desired information are still relevant. This thesis will explore methods and techniques using full-text indexing on next generation *bibliographic information systems* [1], and seeks to propose a search method with an accompanying user model to perform *keyword search* on corresponding datasets.

In comparrison, a *traditional retrieval systems* would often return disorganaized

[1] Systems based on a FRBR-like data models, or others like CRM/FRBRoo, BIBframe etc.

information that will most likely be impossible to present in a reasonable way. Many of them uses techniques like creating large index files on the entire MARC-posts. The final results will contain duplicates and disorganaized information, and the system is unable to relate the search query with the dataset and the current context.

To get a broader overview on the informatin retrieval systems, chapter 2, the theoretical background, will take a look at different information retrieval systems. This will help us get an idea on how they perform and how they structures the data. Later on we will look into ways of searching and traversing data.

1.1 Motivation and Background

The idea of searching through huge databases and texts is not particularly new and innovative. Search engines have made it easy for peoples to find what they are looking for in the "jungle of information", using only simple search terms (keywords). Methods for structuring, processing and retrieving data is getting more and more sophisticated. But this is often not relevant for the users. As a user you only want to type in the words you want to search for, or the topics you want more information on, and get your desired results in return. Although power-users may disagree. Some needs the opportunity filter out parts of the results, having logical operations and advanced search queries. But what if there were tools that made these superflous?

Some of the tools used to perform keyword search only return the most searched keywords, others returns a set of all matches on the given keys. These system are very simple, and might sometimes serve the purpose of the current search system. But when the data is getting more complex, these system will fall short.

There are systems who solves this by being more selective on which entities they indexes. And at the same time structures the data based on some definitions according to the data, and assumtions on the importance of the different entities. The approach in this thesis is not very different from this. It will try to find ways of processing the entities to get the best matches based on a predefined user model.

Keyword Search

Keyword search is a mechanism for retrieving relevant information from a set of documents. It does not require the users to know the structure of the data. Nor having any knowledge about complex query languages. Keyword search will, as mentioned initially, let the user input any search terms, consisting of keywords, to query a large database, and to retrieve the desired information.

By bringing the richness of the bibliographical information all the way to the user interface, we can provide functionalities that would support users in their information seeking process. When doing this we can bring out the value of bibliographic data by providing intuitive search as well as useful presentation and navigation of bibliographic data (Merčun et al., 2012).

Most of previous work on keyword search over graphs finds minimal connected trees that cover all the query keywords. Some have seen that finding subgraphs rather than tress can be more useful for a search system. The closes work to this study is (Kargar and An, 2011). They proposes a way of performing keyword search on graphs by finding r-cliques [2]. This approach differs from a Steiner tree in which case it contains all the connection between the content nodes that is less than or equal to r. A Steiner tree finds a minimal connected

[2] r-clique is a group of content nodes that cover all the input keywords and the distance between each two nodes are less than or equal to r (Kargar and An, 2011)

tree. Chapter 2 will go deeper into topic of keyword search.

1.2 Research Objectives

The main objectives for this study:

- propose a method for structuring and finding information in bibliographical data
- analyse rules that defines a user model
- propose alternative ways to present bibliographical information

1.3 Approach

This study will collect relevant data through literature reviews, and by prototying the theories. The literature will give a broad overview of retrieval systems for bibliographical information, and other information retrieval systems available. It will also contain views and studies on search methods, mostly keyword searching and its relatives.

The result of the literature study will give us a glimpse on the works that have been done, and methods that have been tried out. The data collected will work as the basis for our own search method and its accompanying user model. A detailed explanation on how the prototype was created, and its technologies, can be found in chapter 4.

This study have chosen to work with the following concepts:

- *an FRBR-like datamodel* - a conceptual entity-relationship data model (IFLA Study Group on Functional Requirements for Bibliographic Records, 1998). FRBR is a way of structuring bibliographical information by connecting

the releationship between works and editions of a work. Some have implemented the model, but we have yet to see a solution that can carry out the potential of an FRBR-based catalogue. We will take the data structure proposed by FRBR to restructure and retrieve data.

- *eXist database* - a high-performance native XML database engine. eXist provides XQuery[3], among others, as its query and application programming language. It provides us access to our bibliographical database through a simple interface. By using XQuery we will be able to easily retrieve and work on the data. By using the entity relationships we can easily traverse the relationships between works and editions of works.

- *Apache Lucene* - an information retrieval software library. The eXist database is able to access Lucene to index text that is stored in the database. Our bibliographical records is then full-text indexed by Lucene. This gives us the ability to weight search results by its hit score based on the input keywords by usings its scoring system.

- *Data Visualization* - traditional linear lists do not provide the necessary structure that would be able to display various FRBR-based groupings (Merčun et al., 2012). By using an entity model we will be able to visualize the relationships in the bibliographical database. Our search method will propose a way of filter and hilight relevant results in the result set. With the knowledge of this we can propose a way of presenting the data in a more functional manner.

[3] http://www.w3.org/XML/Query/

1.4 Contributions

The major contributaion of this thesis consist of the data analysis, the search method, and the visualization of various bibliographical information. The analysis will consider multiple search systems and methods to create a better foundation for further work (more about this in chapter 7 (section 7.5). The result will consist of a proposed search method with an accompanying user model that other models can rely on when defining a more generic approach. A simple visualization alternative is given to illustrate a functional way of visualize the data.

User Model

The idea of the *User Model* is to find ways of structuring relations in a dataset. The relationships will of course vary from one dataset to another. To propose a user model for structuring bibliographical data, we have taken a look at the potential of the FRPR entity-relationship model. An FRBR-like model is chosen to create and form our user model. This gives us a dataset containing standalone entities which also holds information about its relationships.

The entities in our user model are: *person*, *work*, *expression* and *manifestation*. This model is then applied to the bibliographical data. In general, the FRBR-like data model enables a better overview of the dataset by encapsulating the information.

Search Method

The *User Model* will act as the foundation of the search method. The intention of search method proposed in this study is to show the joint operation of the datastructure and the keyword search mechanism. Our search method is

very specific to bibliographic data, but can be derived and applied on other user models.

To do so, rules needs to be deinfed in how the relationship is connected and how the entities found in a full-text search would be weighted. The search method is detailed in chapter 3.

2

Theoretical Background

To initiate the study, a collection of relevant literature has been reviewed. The literature is retrieved from, ironically, other online libraries.

Researchers has covered the important questions on the challenges and the complexity of implementing a FRBR-like data model in an online bibliographical library. Not all agrees in how data should be structured, or how it should be processed in order to retrieve the desired results. The results of the different studies contains both pros and cons on how to solve the problem. We are mostly interested in tree based structures and search methods using keywords. Some authors have pointed out problems on tree and graph based methods when performing keyword searches (Kargar and An, 2011). One of the cases was:

> ...while some of the content nodes in the resulting treees or graphs

> are close to each other, there might be content nodes in the result that are far away from each other, meaning that weak relationships among content nodes might exist in the found trees or graphs.
>
> Kargar and An (2011)

There are of course both pros and cons on this one, which depends on the intention of the system. The one case pointed out here is although a great thing in order to present the relevant content to the user. By bundling the weak realationship, the result tree would be complete when it is considered to having all "revelant" information available. On the other hand, this might lead to disorganaized results, and more information to process.

Taking the contrary view, Kargar and An (2011) has proposed to find r-cliques as a new approach to the keyword search problem. The argue that, *"assuming all the keywords are equally imporant, results that contain strong relationships between each pair of content nodes should be preferable over the ones containing weak releationship"*.

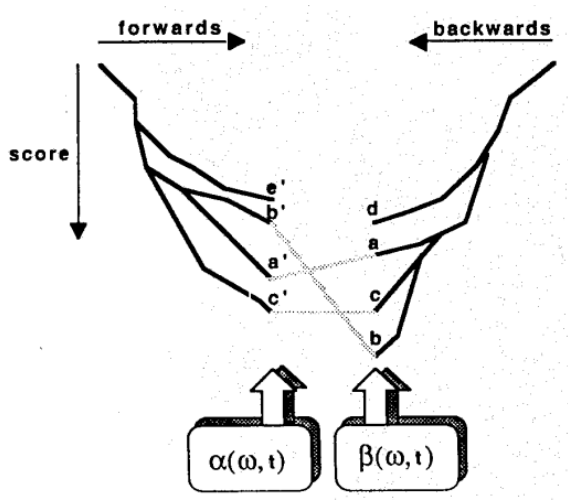

Figure 2.1: Forward-Backward Search. Score on the forward and backward state are added to predict final score for each tehory extension.

Other authors like Bhalotia et al. (2002) and Ding et al. (2007) defines approaches to find minimal connected tress using backward search algorithm and dynamic programming. This methods starts at nodes matching the keywords and works its way up toward confluent roots. An extend to this is to use *Bidirectional Search* (see 2.2.1) which also allows forward search from potential roots (figure 2.1). These systems returns a set of nodes that togheter cover all of the input keywords.

Our Approach

The theoretical background in this study is heavily based on the work by Merčun et al. (2012) - *Visualization of results and navigation support in user interface of Bibliographic Information Systems*, which already utilizes an FRBR-like data model. It derives from the unfulfilled objectives of bibliographic information systems such as online library catalogs, and discusses the challeges library information systems faced by *"repeatedly being characterised as difficult to use, frustrating, and inefficient"* (Merčun et al., 2012). Their study presents issues with current working online libraries, and suggesets how to improve the way one should visualize search results.

The web, and web search, have had a tremendous development, and online library catalogs has not been any worse. The changes on the web have had influence on the users' mental models, their expectations and behaviour when using online library catalogs (Yu and Young, 2004). With all the new search engines like Bing, Google, Yahoo!, and others, users have become more accustomed to searching using natrual language, or keywords, as their search query. These search engines have given users the ability to retrieve information in a fast in intuitive way without having any special knowledge on "what and how".

The direction of online search engines has resulted in users expecting li-

brary catalogs to function as traditional Internet search. The consequence of this is that users finds library catalogs hard to use, unintuitive and ineffective compared to other search sites. But even though users preferred to use the web search over library catalogs, studies have proven that users see the catalogue as more trustworthy, well-organized, and an impressive tool (Yu and Young, 2004). This brings us back to our idea of creating a method for utilizing keyword search over structured data and retrieve well organized and relevant information in, among others, online libraries.

2.1 FRBR

In 1998, K.G. Saur published the final report of the International Federation of Library Associations and Institutions (IFLA) Study Group on the Functional Requirement for Bibliographical Records (FRBR). The report has come to be known as FRBR. The purpose of that study was to

> ... delineate in clearly defined terms the functions performed by the bibliographic record with respect to various media, various applications, and various user needs. The study is to cover the full range of functions for the bibliographic record in its widest sense– i.e., a record that encompasses not only descriptive elements, but access-points (name, title, subject, etc.), other "organizing" elements (classificaton, etc.), and annotations.
>
> IFLA Study Group on Functional Requirements for Bibliographic Records (1998)

The work surrounding FRBR have raised some concerns by users, researchers and developers. FRBR (and other FRBR-like models) and its entities have influenced the way we think of bibliographic data. But to make it possible we need

to understand each component of the model itself. *"Unfortunately, discussions of FRBR sometimes make this difficult by focusing on how the most complicated bibliographic situations fit into FRBR's entity/attribute structure"* (Bowen, 2005). The model itself, and its relatives, might seem limited when considering content and tagging standards, and therefore difficult to understand how it can be implemented in libraries and library applications.

As presented in Patton (2006), the potential benefits of implemeting the FRBR data model in an online catalog are many, including better collocation, more efficient navigation of search results, and better bibliographic control in a global environment. These are the things we want to adopt and achieve in our search method by structuring our dataset in accordance to such entity models.

2.1.1 RDF representations of FRBR

*"Bibliographic information originated by libraries still largely remains buried within the **hidden web**"*. This was one of the observations made by Gradmann (2005) in his work. This research discovered opportunities to carry out RDF-based library catalogs built using the idea of structuring data as model-based entities. This resulted in a proposal for implementing FRBR as a RDF-Schema. Another observation made in his article was:

> ...the sheer amount of data that would probably present major problems when migrating to more generic technical environments prevents most librarians from seriously considering technical and functional alternatives to the current situation.
>
> Gradmann (2005)

The idea, and the potentials of such systems comes with great benefits. But this technique is not very widespread, and questions on how to search and dis-

play the results are still wide open. Finding data, in general, is not a hard task. But finding the "correct data", getting the results you wishes for, is not a trivial task.

As earlier pointed out, these methods might seem complicated and limited, and may explain to some degree why libraries have been so reluctant to seriously consider models such as FRBR as a basis for new librarian information architectures. In the article by Dunsire (2010) he points out that there is still some challenges in creating consistent RDF-labels and definitions based on the source documentation for a FR family. And since we are unable to find great answers to this, it will remain in the dark.

In our study we have chosen to not implement our FRBR-like user model as an RDF-Schema. It could have led to such structuring in a later stage, if possible. But not as a main focus to test the proposal. Such work could be a possible derivation to our work. We will settle with simple entity modelling for now.

The proposal by Gradmann (2005) suggests that *"expressing FRBR in an RDFS model would then allow for implementing catalogs using RDF and for integrating Semantic Web ontologies in such a framework in various fields"*. Which also is a potential derivation of this work. More on this in chapter 7 (section 7.5).

2.2 Keyword search

"Keyword search is a proven, user-friendly way to query HTML documents in the World Wide Web" (Xu and Papakonstantinou, 2005). As one of the gains pointed out by many experts on FRBR, keyword search is one of them (Patton, 2006). Keyword searching have shown to be one of the most effective paradigms for information discovery. Keyword search allows users to find the information they are interested in without having to think of structuring the search query, nor

having to learn a complex query language or have any prior knowledge of the stucture of the underlying data.

As mentioned initially, online search engines have given users the ability to search online documents by entering words they want to search for without having to think of structuring the search query. As these search engines are more exposed to the users in their everyday Internet usage. This method has more or less become the standard for searching on the web.

When performing a keyword search there are two possible ways of expecting the returned resultset:

- returning elements that contains all the keywords in the search term (conjunctive keyword query semantics)
- returning elements that contains *at least one* of the search term (disjunctive keyword query semantics)

2.2.1 Keyword search over graphs

When structuring data for search systems we often model it as graphs, or trees. Keyword search over a graph finds a substructure of the graph containng all or some of the input keywords. Most of previous methods in this area finds connected minimal trees that cover all the query keywords (Bhalotia et al., 2002). A so called *Steiner Tree*. The Steiner tree problem is although a NP-complete problem and cannot be solved in polynomial time, except from some restricted cases. The article by Kargar and An (2011) covers this field by proposing a way of searching through graphcs by finding r-cliques as a new approach to the keyword search problem. *"An r-clique is a set of content nodes that cover all the input keywords and whose shortest distance between each pair of nodes is no larger than r"* (Kargar and An, 2011).

In this study we look for opportunities to relate the work with the idea of initiating keyword search over graphs. The data inside the user model is entity based, and the connection between the entities are their relationships. Together all the entities in the database will form a huge graph. Details about our search method is found in chapter 3.

XRANK

Guo et al. (2003) considers the *"problem of efficiently producing ranked results for keyword search queries over hyperlinked XML documents"* and focuses on the challenges in queries over hierarchical XML documents, in comparrison to flat HTML documents. Problems occurs when the keywords are located deep down in the nested XML elements. The other case is that *"the notion of ranking is no longer at the granularity of a document, but at the granularity of an XML element"*.

Even though our data format is loosely defined as XML-based entities, we find these problem interesting since it may apply for our case as well. XML is used because of its flexibility, which is, in this study, more important than efficiency. This study will not focus on any streamlining of the search process itself.

The idea of XRANK is to consider the importance of an XML element, and not the XML document itself. It is computed based on the hyperlinked structure of XML documents (Guo et al., 2003). It is not unlike Google's *PageRank* (Brin and Page, 1998). The difference is that XRANK computes the granuality of an element and takes the nested structure of XML into account. In our study, this applies to the importance of each entities in the user model.

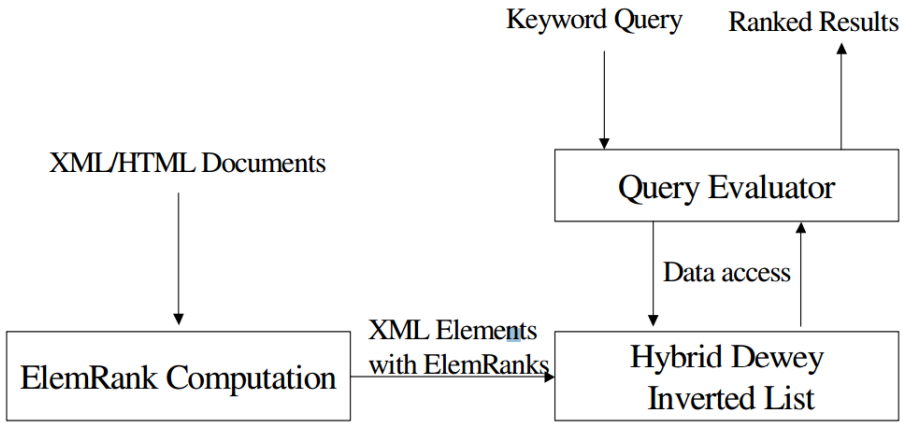

Figure 2.2: XRANK Architecture

Although, the XRANK method depends on a tree-structured XML for efficient query execution. Which means that the XRANK index structure cannot be used in scenarios where data forms arbitrary graphs (Kacholia et al., 2005).

Bidirectional Expansion For Keyword Search on Graph Databases

Kacholia et al. (2005) also takes on the idea of performing keyword search on graphs. Such as text assosiated with nodes and possibly edges. We mentioned *Backward Expanding* and searching earlier. A commonly used method for traversing a search tree. It starts in the nodes matching the input keywords, and works itself up toward confluent roots.

> A central problem in this scenario is to efficiently extract from the data graph a small number of the “best” answer trees.
>
> Kacholia et al. (2005)

Other researchers have imporved upon this method by introducing *Bidirectional Search*. It improves on Backward Expanding by allowing allows potential roots to search for the leaves, while the leaves searches for confluet roots. A

Bidirectional Search algorithm is proposed in a study by Kacholia et al. (2005).

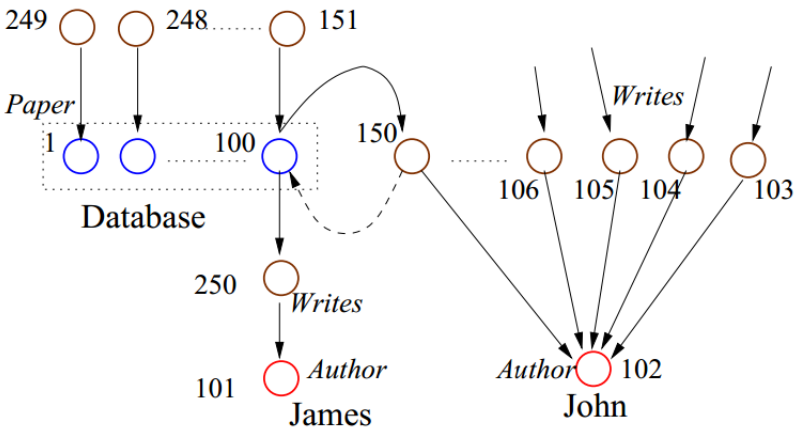

Figure 2.3: Bidirectional Search Example

This thesis proposes a search method using a traversing method not unlike this one. It starts from the top, and works its way down to the leaf nodes. But when run simultaneously from other result nodes, it will get the effect of utilizing a bidirectional expansion when two processes meets in the middle.

DBXplorer

"Internet search engines have popularized the keyword-based search paradigm. While traditional database management systems offer powerful query languages, they do not allow keyword-based search" (Agrawal et al., 2002). DBXplorer prefers trees with fewer edges, and uses the number of edges as a measure of quality (Kacholia et al., 2005). This will not work out well in a context of graph search where node ranks and weights are required. To put it bluntly, DBXplorer will favor a poorly refered data source over a highly refered data source.

Keyword Search over RDF Graphs

Elbassuoni and Blanco (2011) proposed a retrieval model for keyword queries over RDF graphs. Which was the initial idea of our study. Although the way they process the query is to first retrieve a set of subgraphs that matches the query keywords, and then ranks them based on statistical language models. This model have proven that it is able to *"outperform the-state-of-the-art IR and DB models for keyword search over structured data"* (Elbassuoni and Blanco, 2011).

The system is constructed in a way where the knowledge base consist of a set of SPO [1]-triples. Where each triple is a virtual document. When a search is performed, it will generate a list of matching triples for each query keyword. Subgraphs are constructed from these joined triples where each subgraph is unique and maximal. The subgraphs are then ranked based on statistical language-models (Ponte and Croft, 1998).

Instead of using a language model. This study seeks to define its own user model in order to rank and connect the entities.

2.3 Visualization

The following screenshots are provided as examples on how differently online libraries acts when searching with the same search terms. The example also indicated the problem that users have to deal with when receiving large sets of data. The results shows hundreds of elements layed out in a list. Most of them are related, but only a few of them tells the user anything about the related editions. Even those with a grouped results do not really provide a good user interface for nagivating the results.

[1] Subject/Property/Object

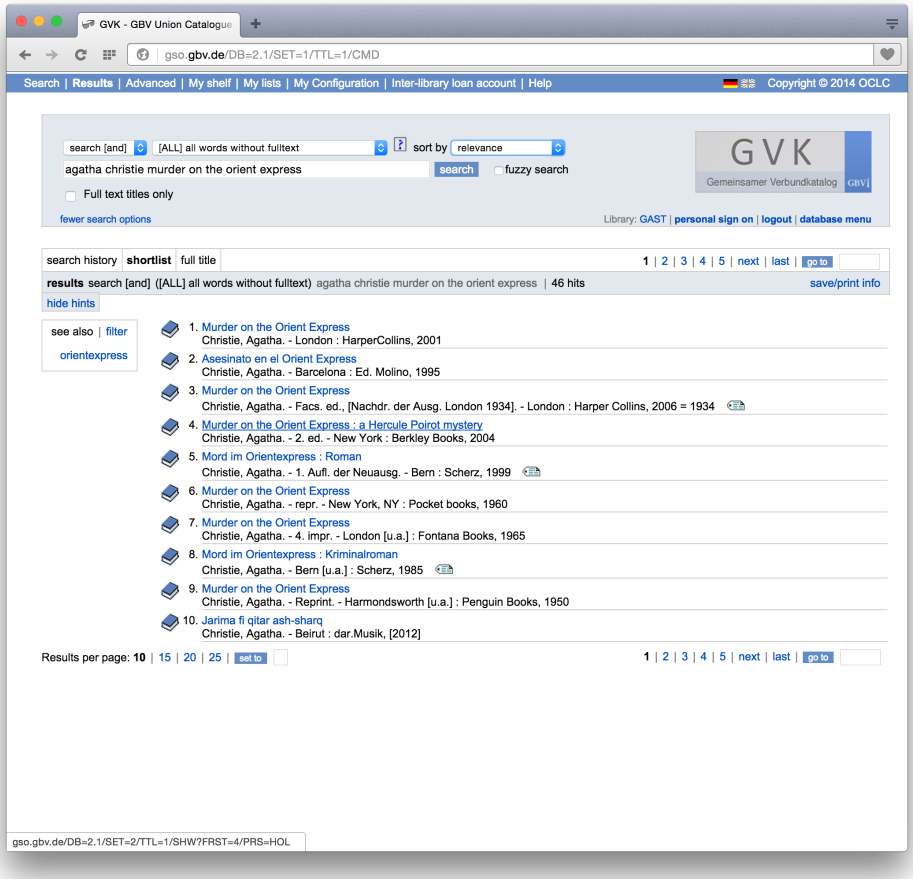

Figure 2.4: Search hits: Gemeinsamer Verbund katalog

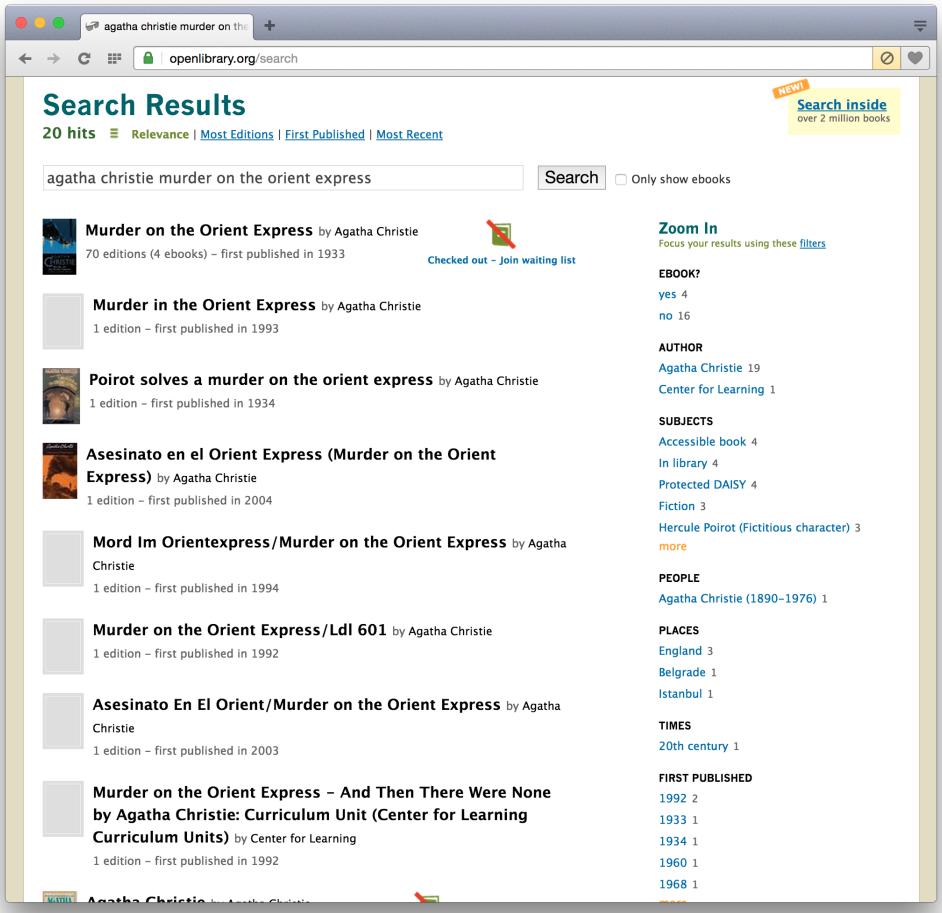

Figure 2.5: Search hits: Open Library

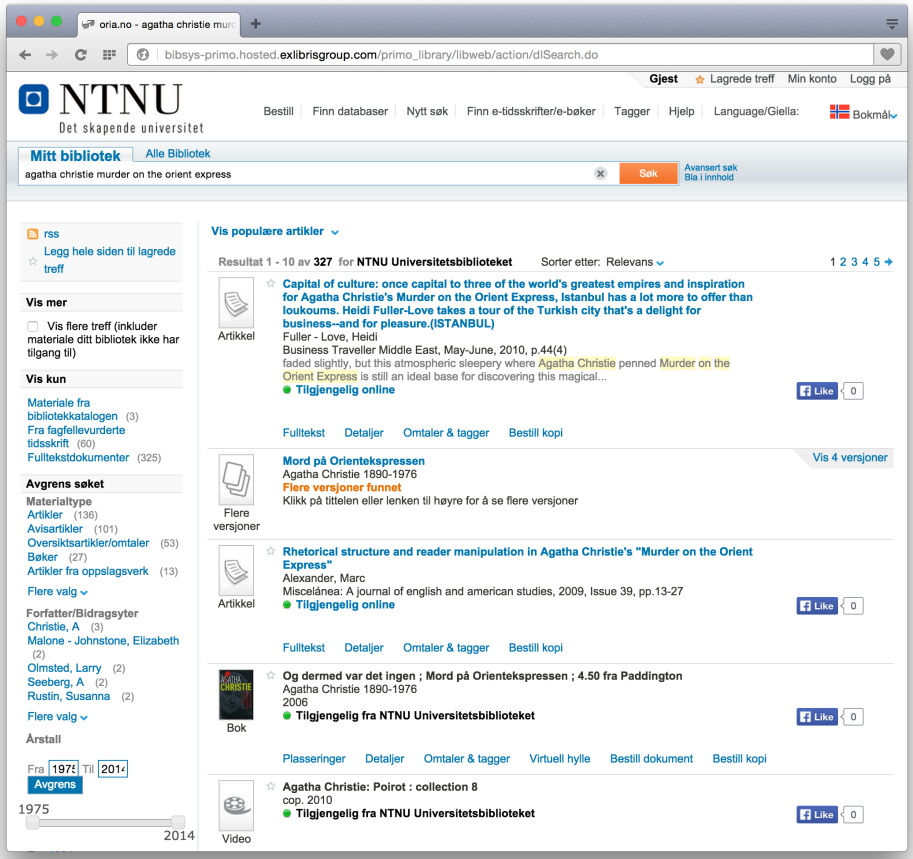

Figure 2.6: Search hits: Bibsys - NTNU Universitetsbiblioteket

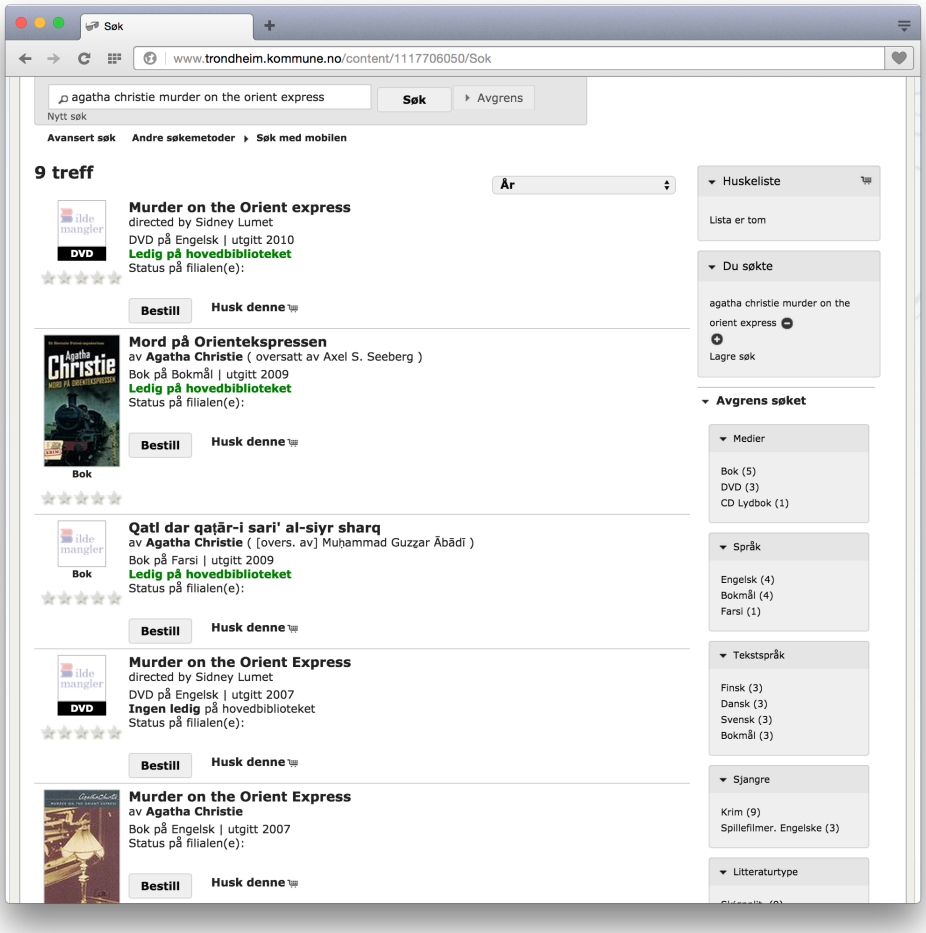

Figure 2.7: Search hits: Trondheim folkebibliotek

Figure 2.4 shows the most traditional way to display results. This example from the *Gemeinsamer Verbund katalog* lays results out in a long list with very limited information. The first ten results of the query *Agatha Christie Murder on the Orient Express* contains multiple editions of the same work.

Figure 2.5 is very similar to 2.4. But this one actually groups the related items, by best effort. Although some of the related results have been left out of the groups. When entering the grouped result, you will get the displayed edition as the main item. The rest, like the format of the works, are listed in a long list with very inconsistent information.

In figure 2.6 we sees the BIBSYS/ORIA approach on displaying the search results. This one is very similar to the one in 2.5, but it has a more consistent way of displaying the work format and its availability. It also gives the users the ability show all the work metadata without navigating away from the results. But also this system lays out all results in a list instead. Just some simple touches to the results could give a better user experience. For example grouping the formats of each editions.

The example in figure 2.7 is very similar to the one in 2.4. This one provides a more cleaner look to the results list, but lacks in connecting the relationships of each work object. This one benefints from having a great overview of the available formats. And at the same time displays the format in a consistent way.

2.3.1 Related Work

Merčun et al. (2012) has done a more detailed study to demonstrate the problems concerning the presentation and navigation in online libraries. They have also created (*FRBR Search (dijon.idi.ntnu.no)*) to demonstrate simple grouped presentation.

From the given examples above. The different online library catalogs still

have different approacheso on how to display and navigate search results. A combination of figure 2.6 and 2.7 would have been a good alternative to the traditional list approach. One observation from these library catalogs, and others, is the problems users have to deal with when facing large sets of results. Some provides the ability to filter out unwanted formats and languages. While other displays all works and editions in a long list, and the users having to to through tens of hundreds of the results to get an overview of the available data.

3

The Search Method

The idea, the definitions, the assumtions, and the choices made to construct the search method is described in this chapter. The process is described from the simple models to the more specific cases.

A previous work on this topic was carried out by (Merčun et al., 2012). Their works was mainly focusing on visualizing and navigation support in Bibliographic Information Systems. The prototype *FRBR Search (dijon.idi.ntnu.no),* a simple search engine were made utilizing the FRBR-like data model. The search method in their work was rather simple in the sense that it did not have any strategy on how to connect the relationships. All relationships to a work was listed underneath the main hits to illustrate the related entitites.

Our search method started out as a theory on how to *construct a subtle data*

model from the documents, with a greater focus on the probability and weighing of each and every relationship. This method seeks to see how a search for specific data will perform when a pre-defined user model is applied to the dataset, as a prerequisite.

In order to do this, rules for each cases, and corner cases, must be defined. A set of rules that think as the user. It needs to assume the importance, and prioritize the relationships. And for each entity it needs to analyze its weight against the others in the particular dataset. For now, it is using bibliographic data, only.

The previous chapter covered some relevant theories and works in the field of keyword searching, and keyword search over graphs. We have also presented some examples on how some of the current online libraries present bibliographical data.

Generalization

This search method seeks to utilize the technique of keyword searching over graphs.

- It starts out with a large dataset. In this case, the dataset contains bibliographic data with works and their authors, and all the related expressions and manifestations.
- This data is then processed and derivated into their respective entities.
- This is stored in a document database which utilize full-text indexing for searching the documents.

The database used in this study is the initialy mentioned eXist database.

In general, to minimize the search graph, the system start out using a full-text index mechanism in the database to retrieve relevant data. The retrieved data is then used to build a search tree according to the defined user model.

When the data structure is constructed, the system can then begin to traverse and process the nodes. This process will apply the rules defined by the proposed search method to get the desired results for the current dataset.

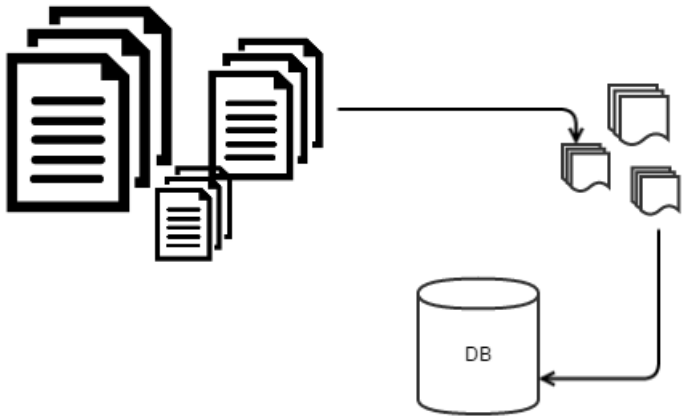

Figure 3.1: Converting from documents to entities

3.1 The Idea Behind the Search Method

The idea of this search method is based on the idea of having the bibliographic information structured as functional requirements for bibliographic records.

This led to the usage of a FRBR-like data model. The initial idea was to use RDF based data. But an early decision was made, which resulted inn constructing simple XML-document to store the most important data, intead of using a RDF-Schema. This decision made it easier to get started trying out the theory.

To conduct search on the constructed data, the system needed a way to receive user queries. This is where keyword search enters the system. More on

the background of keyword searching is found in chapter 2, including (Kargar and An, 2011) who carried out a study on keyword search in graphs by finding r-cliques. While others have been working on minimal connected trees (Ding et al. (2007), Bhalotia et al. (2002)).

This thesis will not go as deep as (Koumenides and Shadbolt, 2014) and (Baeza-Yates et al., 2008), and discuss topics like *Semantic Search*. Neither will topic such as Machine Learning and Natural Language Processing (NLP), which could be an approach when conducting such a study, is not a part of this study.

The work on defining the rules for the user model is somehow simple, but will introduce some complexity when it comes to processing the different kind of relationships. The connections between each entity are prioritized by the definitions. For this to work, we actually need to know what kind of data we are working on. And each type of data needs its own user model.

The well known keyword search method is utilized to receive the data, and further on apply it to our own constructed graphs. This study differs from the ones mentioned above by having a search method that *"knows what it should look for"*.

The process from a keyword query to a result set is defined like this:

1. query the database on its full-text index

2. apply the user model to the returned entities

3. return the structured data in a decending order by its relevance

Figure 3.2: The Process

When the entities are retrieved from the database, the system will then build the relationship trees using the current model that is designed to cover the given type of data. These definitions tells us how relationships are connected, and which connection should be prioritized over the others.

Each entities returned from the database query contains a score from the retrieval process. This score indicates its relevance to the keywords used in the query, and helps us indicating the importance of the entities. Although this is only an indication and may be overrided by a stronger connection. More on this later.

A search on a full-text indexed database, in this case, employs a disjunctive keyword query semantic. And works as follows:

- the search terms are processed by the search mechanism in the database
- the search mechanism will look up the keywords in the generated index file
- for each term we hit (at least one of the keyword - a simple OR query), we will retrieve an accompanying entity
- the entity is then packed with the rest of the results and returned to the middleware

This approach will provide retrieved documents with a similarity scoring to the input keywords. This is a variant of Tf-Idf [1] scoring model. This involves measuring how often a term appears in the document, and how often it appears across the index. It also considers the measure of the importance of a term according to the total number of terms in the field. More on this over at Apache Lucene (2014).

The score allows us to filter out good hits based on their hit scores. The higher score, the better. When applying the relationship rules from the our user model we will be able to distinguish the actual hits from the connected relationships not in the result set.

The first thing the system would do is to prioritize the nodes with the highest score from the retrieval process. Why this is not always the best approach is discussed later. By asuming this, the system may return the data aranged in a decending order by the elements score, having the highest scored node on top, and the lowest at the bottom.

Further on, the search method would connect the relationships. When a branch is created, it will try to find the result branch from the constructed result tree with a highest sum.

[1] Term frequency-inverse document frequency

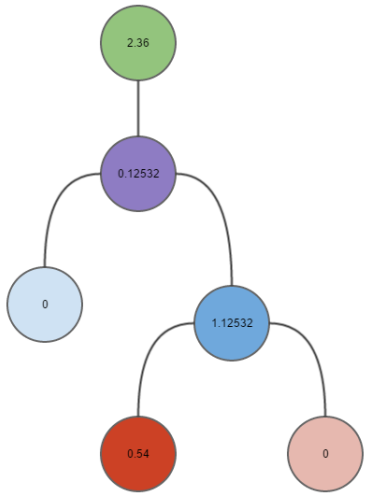

Figure 3.3: Result branch / Search hits

Our proposed user model consists of these four entities:

- *work* - a distinct intellectual or artistic creation
- *person* - an individual
- *expression* - the intellectual or artistic realization of a *work*
- *manifestation* - the physical embodiment of an *expression* of a work

In figure 3.3 the executed search found four nodes that is relevant to the keyword query. In this case it got hits on all four entities (person, work, expression and manifestation). The total score for this result branch is 4,15 *(0.54 + 1.12532 + 0.12532 + 2.36)*. The great thing with this branch is that it also includes the two nodes with the score of 0. These are the nodes that did not get any hits when performing the search, but are directly related to the returned nodes.

One of the challenges when building such result trees from bibliographical nodes is to construct the data structure with all the related entities. An entity

can related to other entities that down the road may depend on the base node. There are many ways of handling a circular relation. Questions like how, and what, will arise. Like how to determine the limit? And what entity should be the root node? More on this later.

The First Rule

The first rule is defined like this: a person entity is a key element, and not a root element.

When a query contains a person, in this case when searching in a bibliographic database, the system asumes that the users wants to see the works done by that person. The focus will therefore rely on the remaining three entities: *work*, *expression* and *manifestation* (figure 3.4). This results in at least three different cases where each one of the entity may be a root element.

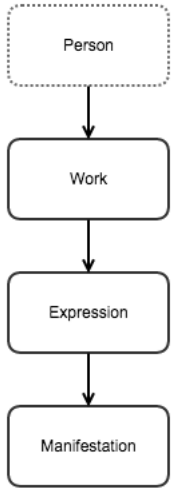

Figure 3.4: The Hierarchy

3.2 Cases

This section will cover some of the most common cases the system would come across, and how it is handled.

What we saw in figure 3.3 was a result branch. A perfect result branch constructed from the full-text search result would contain hits on all of the entities. This means hits on person, work, expression and manifestation. Every result branch needs a root element. How do the system decides its root node? The definition has already stated that: *"a* person *node would not be the root node"* when building the data structure. This case would assign the *work* element in such a branch as the root element

The root node is often the node the system will present for the user. The user interface will of course also contain all its relations, but still having the root node for each branch as the "main hit".

But this is not always the case. There will be cases where the system are unable to build a perfect branch, and only returns some of the nodes, leaving the user with some missing links.

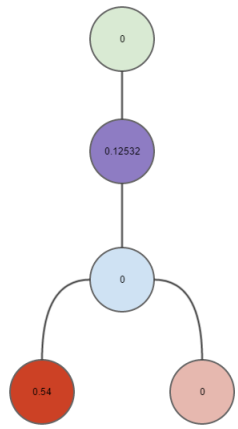

Figure 3.5: Fragmented hits

In such cases, in general, the search method would want to consider the highest node as the root node. The entities are then considered root in a decending order from work to expression to the manifestations. The person entity will, as mentioned initilally, not be considered a root note. Well, no rule without exceptions: *"unless it is the only node"*.

The idea of not having the person as an important node is because of its relevance. It is considered important in the process of constructing the result tree. The person node will help the system consider the relevance of the other entities by their relations.

Although the hit score will not always be the desired measurement when returning the results. In some cases, a work by author A, which derives from a work by author B could be returned, where both nodes are somehow related to each other. And what the user wanted was really the work by author B. This will be covered later in this chapter.

Here is three cases where each of the entity, except person, is used as the root element. These are presented without all the child node relationships.

work as the root node

a *work* has the following relations:

- *person* (*work* → *person*)
- *expression* (*work* → *expression*)

The *expression* in this structure has relations to one or more *manifestations* (*work* → *expression* → *manifestation[s]*). See figure 3.6.

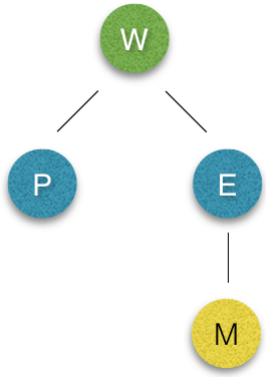

Figure 3.6: *work* as the root node

This would be considered the standard case.

expression as the root node

an *expression* has the following relations:

- *work* (*expression* → *work*)
- *manifestation* (*expression* → *manifestation*)
- *person* (*expression* → *person*)

The *work* in this case also have relation to a *person* (*expression* → *work* → *person*). See figure 3.7.

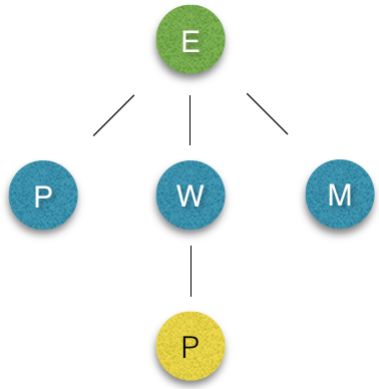

Figure 3.7: *expression* as the root node

The expression entitiy could in many cases have a significantly higher score than the work node. It will then be considered the root node.

manifestation **as the root node**

a *manifestation* has the following relations:

- *person* (*manifestation* → *person*)
- *expression* (*manifestation* → *expression*)
- *work* (*manifestation* → *work*)

The *expression* and *work* nodes in this structure will also contains child items. Respectively:

- *person* (*manifestation* → ***expression*** → ***person***)
- *person* (*manifestation* → ***work*** → ***person***)

See figure 3.8.

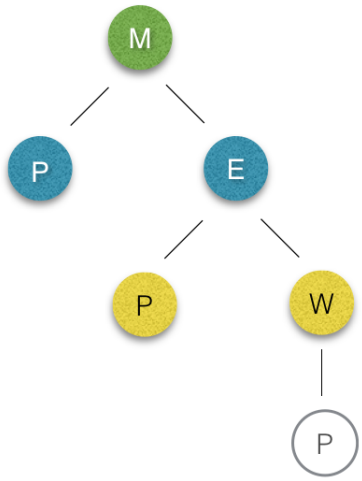

Figure 3.8: *manifestation* as the root node

As with the expressions, a manifestations would sometimes get a significantly higher score than both of the other two entities.

Circular Relationships

Ciruclar relationships may occur as mentioned initially. Which may introduce a lot of complexity when building and visualizing the complete result set as a hierarchy.

To address this issue, the system is constructed to detect the repeating relationship. Which means that it will cut the circular relationship when it finds that the entity already refers to in the current working tree.

In our working search method, the node structure are prioritized according to the entity hierarchy (see figure 3.4). Although the hierarchy may be overridden by rules defined later on. More about the issues can be found in chapter 5.

3.2.1 Search Hits

We have seen that search results varies from system to system. When searching BIBSYS for *"Agatha Christie"* one will get 14 843 hits (checked December 18, 2015). This includes books, videos, articles, CDs, and many more. Within these hits you may find 663 books. If you then narrow down the search by adding more filters to only display books where Agatha Christie herself is the author, then we will get 310 hits. This number is higher than the actuall number of books created by Christi, A. When studying the results you will find duplicated works, and the same works in different languages, as separate result objects.

Our goal is to minimize this duplicated and disorganized information. The search method is designed by default to retrieve distinct values. And will therefore not retrieve the same object multiple times on the same level. Which mean that one entity can not be the root element more than once in the result structure. When multiple entities is related to the same object, they will only contain a reference to an already constructed object.

- TWO AUTHORS - A search query can of course contain all kinds of words and strings. There is no limitation for what a user can type in the search field.

  Although there might be cases where the user actually searches with two names. Like *"Agatha Christie David Suchet"* (Christie, A. and Suchet, D.). Given a case like this, how would the system process the input, and how will it generate the output?

  The first result from this query, without applying any rules, is the person object for *"David Suchet"*:

Listing 3.1: “David Suchet”: object header

```
{
  "id": "3c34d90f-c2ee-3be9-b2b0-2164a5964ce6",
  "type": "person",
  "score": "2.253099"
}
```

The whole object is shown in apendix C.2.

Rules needs to be defined for the system to be concise in every cases like this. A rule may be defined like this: *“it is a high probability that the user is interested in the work the two author have in common”*. In a case where both Agatha Christi and David Suchet are in the same query, the system would find the best match for the two of them. But not neccecerly find a work created by either one of them.

Having the search query *“Agatha Christie David Suchet”* the system will most likely return the work *“Agatha Christie’s Poirot”*, or the *“Murder on the Orient Express: a Hercule Poirot mystery”*. In this case the work is not created by Agatha Christie herself. The two individuals are related through Agatha Christies fictional character Hercule Poirot.

A rule for this case will be as follow:

The work labeled with the *“is a successor (work1) to (work2)”* [2] element (*work1*) is prioritized over the prior *work2*.

Although, in this case

- the only relantionships connected to Davis Suchet is the expression of the *“Murder on the Orient Express: a Hercule Poirot mystery”* which David Suchet is a realizer of

[2] an FRBRer model element

  - ... and the work *Murder in Mesopotamia* which he is an actor in
  - The connection between Agatha Christie and David Suchet, who are the two top results, is the work *"Murder on the Orient Express: a Hercule Poirot mystery"*, when connecting the dots down the graph

- MULTIPLE PERSONS / PERSONS ONLY - The same evaluation of the result set applies for multiple authors like it do in the case where there are two authors. Of course the search mechanism will not know what kind of data the users inputs. All it sees is some keywords. But when the scored data is returned, the entities might indicate what kind of data that was sent to the system.

  But in this case it would be more complex combining and prioritizing their work. Given N authors, there would be at least N works that we need to consider. The best case would be to find the best relation between these authors. This could be done by either finding the work that can be tracked back to as many authors in the result set as possible, or to search for all relation down the relation tree, accessing all types of entities, and do the same calculation. The latter one wille be very resource demanding and might not be considered an alternative at this point.

  Another alternative is to consider the work that is mentioned the most in the relationship tree. If we limit ourself to not consider relationships any lower than to the first manifestation. The system would be less resource demanding, and likely find the *most recurring work*.

- MULTIPLE PERSONS AND NONE RELATED WORKS? - Sometimes the result would return only persons as the top hit. But none of them have any relationships to each other. In these cases, the system is unable to apply

the rules that is defined, and have to revert to an edge case and present the persons work in chronological order instead.

- ONLY EXPRESSIONS ARE FOUND - Sometimes a search query might return only expressions, or the expressions is the majority of the top hits. By top hits we mean the hits with highest scores from the full-text search in the database. A work will not necessarily be the expected result.

  These cases will introduce a more complexed way of visualizing the result. Having an expression as the root node will restructure the way the system think of a work. Although, an expression is a work, in that sense. But is defined as something else.

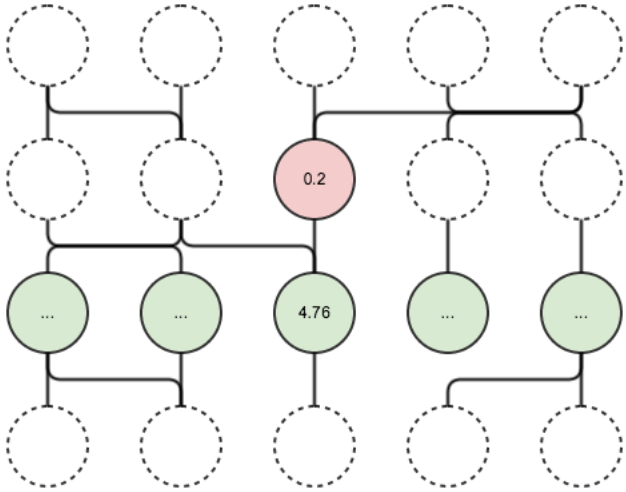

Figure 3.9: Expressions + 1 work

The best thing to consider in such cases are the expressions that we can related back to a *work*. The work node could be a related work to the expression, and not among the search results. The result entities (figure 3.3) can then be prioritized by the work with the most relations to the expres-

sions that are found in the search.

- WHICH ONE IS FIRST? - Sometimes we might get hits on multiple entities with the same type. How would we then prioritize the correct hits? It is easy to think that returning the entities in decending order might be the best way. And it may so be in many cases.

  Sometimes we get result entities with approximately the same score. Given such cases, and we are not able to consider other ways to prioritize the result set, we might want to consider the date of realization (using the FRBR-Element labeled *hasDateOfTheWork* / P3003).

3.2.2 The Result Entities

Each result entity consist of a root node, and its relations. They are prioritized based on the scores of the connected nodes (see figure 3.3).

The illustration above shows that there are cases where branches would not be fulfilled with hits in all levels. No exceptions are made in this case. The system would not justify these hits in any ways by giving it compensation on the missing scores. They will be considered the same as the ones that are retrieved in the first place.

For example. All these cases may be consdered a *"valid result branch"*:

- person → work
- work → expression
- expression → manifestation
- person → () → manifestation
- person → () → expression

- work → () → manifestation

And of course all the other cases, including the single nodes without connections. These are illustrated using the received entities, without the filled in relationships.

Missing Link

As mentioned above, sometimes the system might return search result containing missing links. It might be hit on a work, and some manifestations. Where the expressions are missing. Or maybe work and epxressions, but manifestations are missing, even though they exist as a relation. This is easily explained as the search engine did not get any match on these entities.

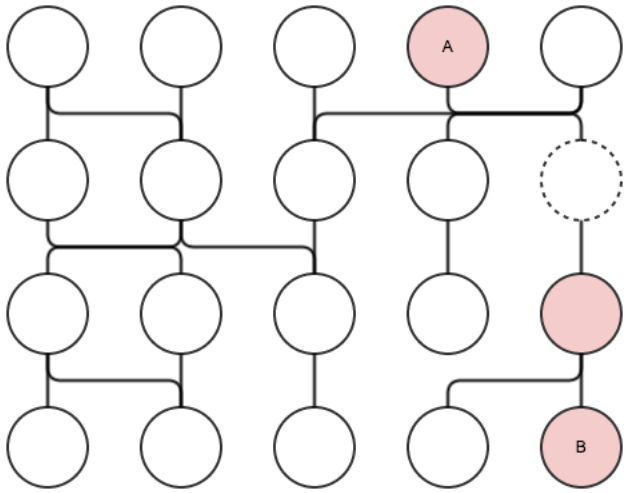

Figure 3.10: Missing node between node A and node B

To handle these cases we will construct the result entity using the nodes that are found, and connects the relations to the nodes that we did not receive from the database. This will give us a complete result entity containing nodes with

the score of 0.

What if a Lower Node Has Significantly Higher Score?

These cases can occur, and we have earlier defined the node structure as work → expression → manifestation. But what if an expression, or a manifestation, has a significantly higher score than its parent? In such cases we need to consider having these nodes as the root node. What the threshold would have to be before we considers this is likely not very easy to define. But using 0.5 and higher as a meassure would give a tolerable fail rate.

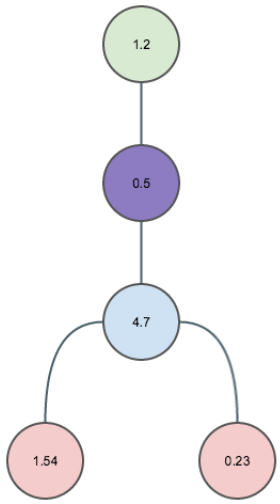

Figure 3.11: The score in the higher nodes is significant lower than one of the lower nodes

3.3 The Search Method Broken Down in Steps

This section will detail the steps from the beginning where a search is requested by the user, to the results returned from the system. It will describe the retrieval process, and how the user model constructs the dataset.

3.3.1 The Search

The previous chapter, chapter 2 (seciton 2.2), described the keyword search and how it functions.

This section will use the concept of traversing graph based on the input keywords to retrieve the desired information. The only interface provided for the user is the search field. Recognizable from other widely used search systems.

The next chapter, chapter 4 (section 4.2.3), will look into the design of the search interface and how it is composed.

3.3.2 The Retrieval

The first step in the retrieval process is to find all matching entities. This is done by running the query through all the entities without any filters. Each entities will return a set of nodes which has their own hit scores. Chapter 4 details the technological implementaion of the method, and how it is carried out.

The Entities

As described initially, our four entities are *work*, *person*, *expression* and *manifestation*. These entities are connected through their relationship definitions.

- a work is connected to a person throug the *creator of* relation in a person. Or the other way, a work is connected to a person with the *created by* relation.

- a work can be realized through an expression. This is marked with the *is realized through* relation on the work. And the other way, an expression is a realization of a work through the *is realization of* relation.

A returned entity might look like this:

Listing 3.2: “Agatha Christie”: object header

```
{
  "id": "a6a32b20-ff01-353c-bc10-604ed3d33b0c",
  "type": "person",
  "score": "2.5855842",
}
```

In this case, the entity contains a person represented by its id, the type of the entity, and the score gotten from the search engine. Further on, such result might also contain complementary data such as name and other useful metadata:

Listing 3.3: “Agatha Christie”: subfields

```
{
  "subfield": [
    {
      "code": "a",
      "type": "P3039",
      "#text": "Christie, Agatha"
    },
    {
      "code": "d",
      "type": "P3040",
      "#text": "1890-1976"
    }
  ]
}
```

This sublist can possibly contain tens of hundreds of elements, each one with its type and content. In this example you can see two emelemnts of different types:

- *P3039* represents *has name of person*
- *P3040* represents *has dates of person*

These element types are the types from the FRBRer data model (Registry). These are also used to evaluate and prioritize the importance of the entities, like the case mentioned above about searching for multiple authors.

The next and likewise important part is the relationships. These are tagged with a type (the relationship type) and its ID.

Listing 3.4: “Agatha Christie”: relationships


```
{
  "type": "P2010",
  "subtype": "aut",
  "href": "099428a4-e126-3430-88b8-38c9e3c38c49"
},
{
  "type": "P2010",
  "subtype": "aut",
  "href": "56a9653c-a104-3738-88ea-8b0cd58ad1e5"
}
```


In this example, the type is *P2010*. This means that this relationships is *created by* its parent. Which is the retrived object of *Agatha Christie*.

The whole object quen querying Agatha Christie is found in Apendix C.1.

3.3.3 The Construction of the Result Set

The data retrieved is then carried on to the factory where the dataset is constructed. In this step we will apply everything we have gone through so far. This step will traverse the result set from the database, and prioritize the nodes from what we have learned.

This is the step where the user model is applied. A user model containing rules and other definition to prioritize the right entities for the best possible results. And this is the process of returning the data to the user.

The Result Set

The result set is a linked list containing the most valued hits and its relations. By connecting all its relations, both the one that was found during the search and the those that were not retrieved, we can serve the users with the desired data,

and the relatd data.

What we mean by *the one that were not retrieved* is all the nodes that did not get retrieved in the full-text search, but are directly related to one of the resulting entities. These are nodes that can provide the user with more information on the returning resultset.

We mentioned earlier that duplicated retrieval would not happen. And this is true to a certain extent. Duplicated object would not appear from the system. But it is still able to interpret the results as duplicated since the same IDs, the identification of an entity, are returned multiple times. What this means is that the object is returned only once, but inside all of the relationships, the same ID might occurs in multiple entities. Mostly as relationships.

3.4 Summary

To sum it all up. The search method is a set of rules that are connected to find the best matches. The rules are defined in the user model which is tailored for each and every types of datasets. There are some uncovered weakness in the system that is listed up in chapter 5.

4

Implementation

As described in chapter 1, parts of the thesis derives from the field of keyword searcing to explore possibilities of using pre-defined user models to structure and find data. A prototype of the search method is carried out to test and retrieve the information stored in the database. It functions as an interface for retrieving the entities and its relations.

The previous chapter detailed the search method to be implemented. This chapter will describe the prototype designs and technology. We will take a closer look at its implementation, and its inspiration which we introduced in chapter 2. And futher on we will discuss the technologies that is used in the prototype.

4.1 Designing and building the prototype

A broader overview of the idea and functionality of the prototype is found in chapter 3, and its background in chapter 2. This prototype will reflect the assumtion we made upon the user model in chapter 3, and is one of the key points for structuring the system.

When designing this system, we decied to implement it without taking any system optimization into account. The search method itself is also not designed for best performance with time complexity and system resoruces in mind. This prototype togehter with the search method proposed will only be a proof-of-concept to demonstrate the *search method*. This lets us experiemnt with the data and tweak the search method to the proposed user model.

4.1.1 Inspirations

The main inspiration for the *Prototype* is based on past work done on at *FRBR Search (dijon.idi.ntnu.no). FRBR Search (dijon.idi.ntnu.no)* mainly searches for all the model entities (persons, work, expressions and manifestations). The distinct results are then connected through its relationships. Although this systems does not take scoring and entity interpretation with rankings in mind into account.

Our prototype is heavily inspired by its data presentation, and how it is querying data behind the scene.

4.2 Technology Stack

This section will abstract away certain elements that underlies the system. We will not consider any technical details when it comes to server hardware, nor security ot network setup. The main focus is on the usage of the document

database, the implementation of the middleware, and lastly the user interface. We will also introduce the technologies used, but in the sense that we only skims the surface. There are better places to read about the technologies.

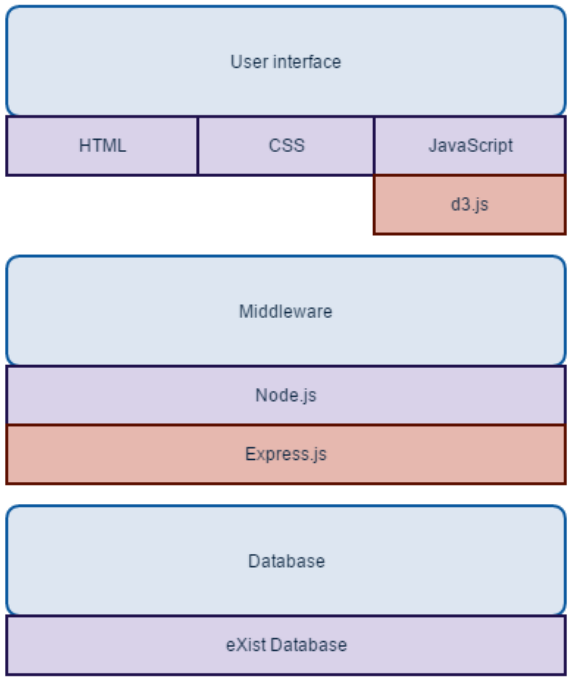

Figure 4.1: The technology stack

As shown in figure 4.1 the prototype contains two main parts. The backend with all the search logic, and the front end with the design principals. The back-end is also devided into two parts. The lowest part is the database which can be located locally, or remotely. The other part is the middleware, which is also the logical part.

The front layer of this prototype is responsible for performing the query from a user against the middleware, and to present the data for the user.

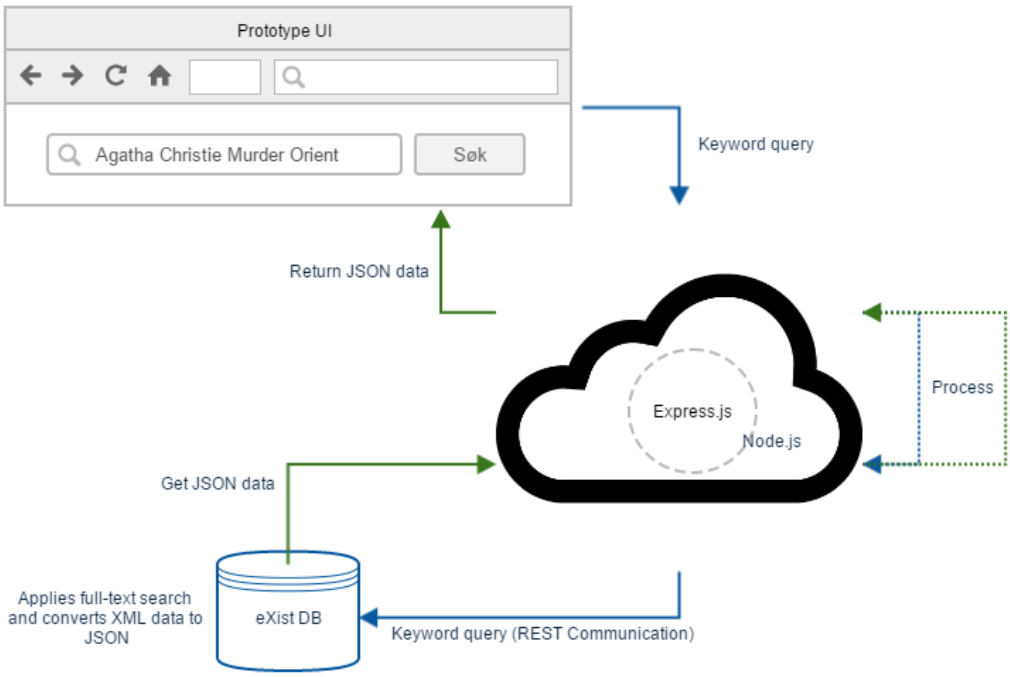

Figure 4.2: Architecture at a glance

4.2.1 The database

The Prototype is designed and developed on top of an eXist database to store the XML-files. A NoSQL document database built on XML technologies which uses Lucene to index documents. eXist then offers an interface for communication between the application and the database. eXist was introduced in section 1.3

The bibliographic data used in this study is first converted to XML[1] using a FRBR-like data model. The entities, which is this case are the documents, are stored and indexed inside the database for easy access and lookup.

[1] See acronyms

Indexing

The documents in the database acts, as previously mentioned, as one separate entity, and is indexed by their records. Since we use eXist as our document database, we will hae Lucene easily available as our text search enginge. Lucenes text search employs disjunctive keyword query semantics (OR query).

When Lucene indexes a document it breaks it down into numbers of terms. The term is associated with the documents that contains it, and stored in an index file. Such an index file allows easy access to any records given its key.

When carrying out the search query query. The database analyses the keywords from the query the same way as it did when indexing the documents, and looks up the matching terms. When all the matching documents are found, it will return it to the unit who requested it. We then combine all the entities based on their hit score from the search result. More on how the search method works, and how the entities are ranked, can be found in chapter 3.

4.2.2 The Middleware

The primary function of the middleware is to process all incoming data, and pass it between the service and the users. It functions as a duck-tape between the database and the user interface. Its three main functions are:

- request data from the database
- process the retrieved data
- serve it to the user interface

By receiving a keyword query from the users, the middleware will pass it through to the Lucene search engine in the database. The information from the database is then retrieved back to the middleware where it is processed. The

final results are then returned to the users. More on the search system itself is found in chapter 3.

The main technology that is used in the middleware is Node.js. With Node.js as a base, we build the whole system on top of the express web framework.

Node.js / express

Node.js® is a platform built on the Chrome JavaScript runtime for easily building fast and scalable network applications (Node.js, 2014). Node.js lets us build service-side appliaction using JavaScript. Its speed and flexibility pave the way for rapid development. By introducing a full-stack JavaScript prototype we are able to reuse components and resources throught the project. One will also benefit from idea of only have to work on the one tech, and not having to maintain code in different languages.

Express is used as the web framework on top of Node.js. It handles all HTTP requests, and lets us create and serve data from its simple APIs. Express gives us the opportunity to collocate the server code and the user interface, but yet sepparates it into their logical locations. As mentioned initially, we build the whole system on top of the express web framework.

4.2.3 The Interface

"We're often told that we should design our websites and software to mimic real-life objects" (Arment, M, 2010). But in our case we would rather not mimic the old analog bibliographic records, nor the en physical encyclopedia as a metaphor for information seeking. Despite a real-world object would be more recognizable for a user, it will not give us any room for serving information in a more interactive manner.

Since we have chosen to not use a real-world object as a mataphor for the

design of our application, we are able to apply more functionality, and at the same time offer more dynamic use and feel. It is important to think of the usability on the platform present for the system and therefor find a balance in the real world reproduction when designing the system.

Technology

Technologies used here is plain HTML, CSS and JavaScript. We are creating it as simple as possible, but yet functional, without having to depend on huge demanding frameworks. The main reason for not including other frameworks into the prototype is to reduce the code base and the overhead it brings to the system.

The Search Field

The interface provides a very simple search field. The search field is likely very recognizable by not being anything else but a traditional search field users are known to in their prefered everyday search engines.

We can start thinking of this prototype as a system for looking up index cards. The users wants to find the data they are looking for, and the system is the librarian who looks it up. The request from the user is handed to the librarian through this search field. When thinking of the methaphors used in user interface designs (Arment, M, 2010), we are not trying to reproduce the elegant encyclopedia, nor the old index cards from the libraries as we know them. It is implemented in a conventient way for the users to use their input devices to simply ask the system for the information.

4.2.4 The Navigation

The user interface of the overall application is created to have a flat information structure and does not contains any advanced navigation to get to the destination.

> In a hierarchical app, users navigate by making one choice per screen until they reach their destination.
>
> Apple Inc. (2010)

Although we are combining the hierarchy in the matter of using a hierarchy approach on the search results. This let the users navigate through a relationship tree without having to leave the page, or reload the search.

4.3 Limitations

The used methods limits the general applicability of the study.

- The code is created specific for one type of data, and is customized to apply only to its structure and content. This means that other datasets needs a whole new code base to function.
- The amount of data is limited. This point is related to the first. By using only one static, well known, dataset, you trick youself into thinking only for this one case. Although new content have been added to the dataset during the study, and patches has been done to support that, it will not give other data types the opportunity to easily adopt it.
- Resources and capacity - having a study that only spans over 10 month, and at the same time cover multiple data sources with limited resources is both time and energy consuming. Although there is no reason to blame

the lack of computer resources in this case, it would be required when wanting to try out such systems in a larger scale.

Other kinds of limitations and problems related to the implementation and the search method itself is found in chapter 5.

5

Problems in the *Search Method*

Problems in, and related to, the search method are discussed in this chapter. These problems range from simple construction errors to more complex question as to wether it fulfills its intended role. The section are looseley organized from going through some simple errors, to covering the more complex ones.

5.1 Problems with the technology

Some researchers have raised some concerns on the data model we depends upon in this thesis. Many of them are only concerned about its complexity, and the work that have to be done in order to fullfill the implementation (Bowen, 2005).

5.2 Problems with the implementation

The implementation of the search method is very simple — conducting simple tree traversal. Optimization are not considered an important part of the study. Due to this, the search method is more or less very slow, and not optimized for production.

- The hard fact would not always come from the score given by the full-text search mechanism in the database. This search method depends on a vendor score. Although the Apache Lucene scoring is blazingly fast, and hides almost all of the complexity from the user.

  This scoring depends on how the data is indexed. Would another approach on how the data is indexed help the search method becoming more effective, or more precise?

- Problems occurs when other than work entities are rated highest in the result set. How would the system interpret the importance of the nodes, and how would it build a result tree? In some works, a related element could be more valuable than others. Maybe one of the expression of a work is historical more appreciated than the “work itself”.

  Later research on this might find it statically possible to be decided by taking all the relationships into account. But this study did not cover these cases. So the answer would be left unanswered until then.

- When defining the person as a helper node, and not a primary node for a result branch. How would the system consider the work by an author if the search term only includes the atuhors name? The theory of this study do not cover this case.

In many scenarios, this would be a problem for the user. The user actually needs to be more specific in the search query. Some might think the data needs some kind of relevance scoring. A question that arise in a setting like that could be: how do we define the relevance of a work? This is related to the problem described in the previous point.

- Circular relationships may occur. This is a returning problem. Although some queries will not trigger this problem. But the more detailed ones may trigger it 10 out of 10 times. This is because a huge amount of entities are returned as relevant nodes to the search query.

  This study conduct a very simple solution by cutting the dependenciey when detecting a returning node in the current working tree. This will not consider a tree built from other nodes. By considering other possible result trees from the current working nodes and its relationships, a result set might look totaly different, and maybe having better results.

- Priorities when scores are simmilar. What data model element would have the greater priority? What makes one element more important than the others? Is there a general approach on this?

- Chapter 3 described the cases where only persons are returned from the database. And the case where none of the returning results had works related to each other. In such cases, what is the prefered approach in constructing the result graph for the user?

  This study returns the works by the listed author in a chronological order without consider any weighing on the entitites. And will not consider the content of the entities.

5.3 Summary and Conclusions

Our approach for finding information have led to some unanswered questions. It introduces the full-text search engine, Apache Lucene, a user model using functional requirements, and a search method having to choose its path through the graphs. These question will later be presented in chapter 7.

6

Process evaluation

This chapter will summarize the process of the study, and we will see how it was carried out. We will try to sum up what have done, and what we achieved.

Our study is very focused on the study by Merčun et al. (2012) - Visualization of Results and Navigation Support in User Interfaces of Bibliographic Information Systems. They started out looking at ways of structuring and displaying the data. Our work is more or less a projection of their work, with greater focus on the search method itself.

6.1 Theoretical background

The goal for the literature review was to build a theoretical background for the proposed search method. After defining the problems of which the study was

going to research, the litterature review was carried out to gather as much information as possible related to our problem. We wanted to see research on the different mechanism we had planned for our study, not neccecerely finding only the studies which covered all of the topics. But rather the ones who detailed parts of it.

The main topics we wanted to research was:

- the trends in information retrieval systems
- usage of FR-like data models like FRBRoo, BIBframe etc.
- keyword search - this part also covered keyword search over graphs

And other interesting and relevant topics like:

- information retrieval systems (traditional systems versus library catalogs)
- presentation of bibliographic information

Newer research and general articles were preferred. Although there was some background information that went right back to where it began. The theoretical background can be found in chapter 2.

6.2 Building the prototype

The process towards this thesis was mainly focused on the prototype. It was built using plain Java, without any framework from third party vendors. This got us up running in time to test how it would be to receive data on a platform not being the search engine itself.

Later on we implemented a REST API using the Play Framework (Play, 2014) to serve the data over HTTP. A great framework for having a RESTful service up running fast.

The comprehensive documentation surrounding Java and its associated libraries was the main reason for going with this technology. But Java turned out to be very slow in development for our purpose.

6.3 Change of technology

After the first evaluation of the prototype written in Java, using frameworks and libraries to support our functionality and development, we had to look towards other solutions. The development time combined with its performance led us to having to re-implement it in another language for faster data and network processing. Although we decided not to focus on performance, having a faster system helped us focus on the method itself, instead of having to tweak the codebase.

The first version in Java had some bottlenecks when reading and constructing the data structures. It consumed a whole lot of time just to test a query. So we had to think different. A "new" prototype was built using node (Node.js, 2014). This gave it a huge performance improvement.

6.4 Desining the user interface

The user interface was mostly based on the work and research by Merčun et al. (2012). Their work covered a lot of the problem that exists in current online libraries, and had some theories on how to improve it. In section 2 of Merčun et al. (2012) multiple online libraries are considere to propose *"a model for improving the presentation of bibliographic records and navigation within bibliographic information systems."*.

The prototype was meant to be implement as simple as possible to simplify the data visualization. But due to some technical problems and its time con-

sumtions, we only imlemented the important parts of the search method, instead of focusing on the user interface.

More on the interface can be found in chapter 2.

7

Conclusions

This study was set out to find and evaluate ways of structuring and searching in bibliographic catalogs with models heavily dependend on FRBR-like conceptual entity-relationship model. This chapter vil conclude our study by providing further discussions on our method used for searching and structuring, its limitations, and issues during the process.

We have already detailed the impementation of the prototype, and seen how it works in a more visual manner. Suggestions and recommendtation for future research and work on the prototype will be presented at the end of this chapter.

7.1 The Search

A search method was proposed with an accompanying user model for the given dataset. We have seen it in action, and how it carries out the result from the underlying database.

7.1.1 The User Model

The user model plays an important part of the search method for it to function. As mentioned initially, an accompanying user model to the search method was proposed. It is heavily inspired by a FRBR-like data model, which introduced a flexibility to shape and mold a user model for our purpose. This have given us the opportunity to easily manage and maintain the data we have.

Other approaches could have been considered, but this was the one we believed in after seing it in action in Merčun et al. (2012).

More on the search method id found in chapter 3.

7.2 The Dataset

It started out with a dataset, that was later on updated with approximately 2000 more entities. This led to some minor issues with huge dependency stacks which we did not cover in the early stage. With some adjustments, everything was up running smoothly agian.

Later on, some adjustments was made to decrease the complexity and size of the dataset. This includes removal of some of the namespaces and the type URI [1].

[1] Uniform Resource Identifier

7.2.1 Data Limitations

One of the limitation of the study was the dataset. The designed user model was only for one type of data. The best way of qualifying the theory would be designing multiple user models for multiple datasets, and then combined the results in each one of them to see their performance.

7.3 The Interface

The user interface in this study did not come through as a complete product. But the essential parts of giving the user an opportunity and functionality to discover desired search results, and the all related data, is presented through the search results.

More discussions on result visualization and navigation is found the study done by Merčun et al. (2012).

7.4 The Technology

This study did not measure the systems performance in terms of speed and resource used. The techonolgy used was the technologies the author of this paper were familiar with. The only requirement for technology was to have it up running in short time.

For the future, technologies might need to be covered in a research to see what would be the best approach for solving these kinds of problems. This goes from the nework and the distributed systems, to the database, the logical part, and the user interface.

7.5 Future Research

In this study we have looked at search techniques over graphs, and keyword search in general. We have proposed a search method using a pre-defined user model on the particular dataset. This section lists questions that, hopefuly, future research will answer. To avoid doing research of future research, there will not be any detailed study, nor any farther classifications on the topics.

- When using a well defined FRBR-like conceptual entity-relationship model. Is there any way of constructing a generic user model? Is there any other alternative data models that make the process more effective?
- Would a top-down approach help the user in finding its desired information? Or vice versa using a bottom-up approach. Would the system benefit from using a RDF-schema on the entities?
- After conducting a full-text search. What would be the most optimized way to construct the tree, and to traverse it? How can the search method find the same results in a more effective way?
- How will an implementation of this search system, using this kind of data models, apply to a current database in production?
- Can the problems described in chapter 5 be fixed?
- Knowing what we know today, what would the ideal search system look like?

7.6 Summary

Future research should cover a broader overview on keyword searching using in graphs. The next important step in this study is to test the method in a more

general manner using different types of data. Studies have to see alternative ways of optimizing the structuring step, and the graph traversing step.

Searching is a hot topic, and have much room left for future research. It might seem easier to design a system where it can assume what the user wants based on the information that is stored in the database, if the system only contains one kind of data. The idea is discuseed, but needs more generalization to make a generic approach to the problem.Seciton 7.5 has posed some questions which may be of interest for researchers in the future.

A

Acronyms

API Application programming interface

CSS Cascading Style Sheets

FRBR Functional Requirements for Bibliographic Records

HTML Hypertext Markup Language

HTTP Hypertext transfer protocol

IFLA International Federation of Library Associations and Institutions

NTNU Norwegian University of Science and Technology

RDF Resource Description Framework

REST Representational state transfer

SPO Subject, Property, Object

URI Uniform Resource Identifier

XML Extensible Markup Language

B

Additional Information

B.1 Visualization examples

The visualization examples used in chapter 2 are four different online libraries:

- Figure 2.4 - Gemeinsamer Verbund katalog

  http://gso.gbv.de/

- Figure 2.5 - NTNU Universitetsbiblioteket (BIBSYS[1])

  http://ntnu.no/ub

- Figure 2.6 - Trondheim Folkebibliotek

[1] A supplier of library and information systems for all the Norwegian university and college libraries

http://trondheim.kommune.no/folkebiblioteket/

- Figure 2.7 - Open Library http://openlibrary.org

Bibliography

Aalberg, T. and Merčun, T. (2014). Frbr search. Accessed november 28, 2014.

Agrawal, S., Chaudhuri, S., and Das, G. (2002). Dbxplorer: a system for keyword-based search over relational databases. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 5–16.

Apache Lucene (2014). Lucene Scoring - TF/IDF. Accessed: 15 December 2014.

Apple Inc. (2010). ios human interface guidelines – platform characteristics - navigation. Accessed november 28, 2014.

Arment, M (2010). Overdoing the interface metaphor. Accessed: 01 June 2013.

Baeza-Yates, R., Ciaramita, M., Mika, P., and Zaragoza, H. (2008). Towards semantic search. In Kapetanios, E., Sugumaran, V., and Spiliopoulou, M., editors, *Natural Language and Information Systems*, volume 5039 of *Lecture Notes in Computer Science*, pages 4–11. Springer Berlin Heidelberg.

Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., and Sudarshan, S. (2002). Keyword searching and browsing in databases using banks. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 431–440.

Bowen, J. (2005). Frbr: Coming soon to your library? Accessed Nov 25, 2014.

Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks and {ISDN} Systems*, 30(1–7):107 – 117. Proceedings of the Seventh International World Wide Web Conference.

Ding, B., Xu Yu, J., Wang, S., Qin, L., Zhang, X., and Lin, X. (2007). Finding top-k min-cost connected trees in databases. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 836–845.

Dunsire, G. (2010). Interoperability and semantics in rdf representations of frbr, frad and frsad. In *Concepts in Context: Proceedings of the Cologne Conference on Interoperability and Semantics in Knowledge Organization*, page 113.

Elbassuoni, S. and Blanco, R. (2011). Keyword search over rdf graphs. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 237–242, New York, NY, USA. ACM.

Gradmann, S. (2005). rdfs:frbr–towards an implementation model for library catalogs using semantic web technology. *Cataloging and Classification Quarterly*, 39(3-4):63–75.

Guo, L., Shao, F., Botev, C., and Shanmugasundaram, J. (2003). Xrank: Ranked keyword search over xml documents. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 16–27, New York, NY, USA. ACM. Accessed Nov 25, 2014.

IFLA Study Group on Functional Requirements for Bibliographic Records (1998). Functional requirements for bibliographic records : final report / ifla study group on the functional requirements for bibliographic records ; approved by the standing committee of the ifla section on cataloguing. Accessed Nov. 25, 2014.

Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., and Karambelkar, H. (2005). Bidirectional expansion for keyword search on graph databases. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 505–516. VLDB Endowment.

Kargar, M. and An, A. (2011). Keyword search in graphcs: Finding r-cliques. *Proceedings of the VLDB Endowment*, 4:681–692.

Koumenides, C. L. and Shadbolt, N. R. (2014). Ranking methods for entity-oriented semantic web search. *Journal of the Association for Information Science and Technology*, 65(6):1091–1106.

Merčun, T., Žumer, M., and Aalberg, T. (2012). *Visualization of Results and Navigation Support in User Interfaces of Bibliographic Information Systems*. T. Merčun.

Node.js (2014). Node.js.

Patton, G. (2006). What can frbr do for you? Accessed Nov 25, 2014.

Play (2014). Play. Accessed: 15 December 2014.

Ponte, J. M. and Croft, W. B. (1998). A language modeling approach to information retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pages 275–281, New York, NY, USA. ACM.

Registry, O. M. Open metadata registry. January 15, 2015.

Xu, Y. and Papakonstantinou, Y. (2005). Efficient keyword search for smallest lcas in xml databases. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 527–538, New York, NY, USA. ACM.

Yu, H. and Young, M. (2004). An impact of web search engines on subject searching in opac. *Information Technology and Libraries*, 23.

C Code

C.1 Enclosed ZIP Archive

This section descrbies the contents of the archive enclosed with this thesis. The archive is available as an attachment.

The structure are as follows:

- existdb-node/ A folder containing the logic for connecting to an eXist dabtabase. This is a patched version of the one found on http://npmjs.org/ which lacks features when querying information from the database
- lib/ The main folder for the code. This folder contains all the logical files for the search. Both helper class and the app itself. *app.js* is the main file for the search applicaiton.

- public/ This folder contains all the CSS, JavaScript and data files that is served to the users visiting the search system through a browser.
- views/ These are the views that is displayed in the users browser.
- xqs/ These are the XQuery files used to query the remote database
- config.json (file) This file contains the credentials for the remote database

C.2 Entities

The object for “Agatha Christie”. Without the connected relationships.

Listing C.1: “Agatha Christie”: object

```
{
  "id": "a6a32b20-ff01-353c-bc10-604ed3d33b0c",
  "name": "a6a32b20-ff01-353c-bc10-604ed3d33b0c",
  "type": "person",
  "score": "3.6659446",
  "doc": {
      "id": "a6a32b20-ff01-353c-bc10-604ed3d33b0c",
      "type": "C1005",
      "datafield": [
          {
              "tag": "100",
              "ind1": "1",
              "ind2": " ",
              "subfield": [
                  {
                      "code": "a",
                      "type": "P3039",
                      "#text": "Christie, Agatha"
                  },
                  {
                      "code": "d",
                      "type": "P3040",
```

```
                    "#text": "1890-1976"
                }
            ]
        },
        {
            "tag": "700",
            "ind1": "1",
            "ind2": "8",
            "subfield": [
                {
                    "code": "4",
                    "type": "marc:M700184",
                    "#text": "aut"
                },
                {
                    "code": "a",
                    "type": "P3039",
                    "#text": "Christie, Agatha,"
                },
                {
                    "code": "d",
                    "type": "P3040",
                    "#text": "1890-1976."
                }
            ]
        },
        {
            "tag": "700",
            "ind1": "1",
            "ind2": "2",
            "subfield": [
                {
                    "code": "4",
                    "type": "marc:M700124",
                    "#text": "aut"
                },
                {
                    "code": "a",
                    "type": "P3039",
                    "#text": "Christie, Agatha,"
```

```
                    },
                    {
                        "code": "d",
                        "type": "P3040",
                        "#text": "1890-1976."
                    }
                ]
            }
        ],
        "relationship": [
            {
                "type": "P2010",
                "subtype": "aut",
                "href": "099428a4-e126-3430-88b8-38c9e3c38c49"
            },
            {
                "type": "P2010",
                "subtype": "aut",
                "href": "56a9653c-a104-3738-88ea-8b0cd58ad1e5"
            },
            {
                "type": "P2010",
                "subtype": "aut",
                "href": "506b427a-9db8-3417-871b-13cb5bc0ce54"
            },
            {
                "type": "P2010",
                "subtype": "aut",
                "href": "62137baf-4509-36b8-bbc0-2b4cb0320704"
            },
            {
                "type": "P2010",
                "subtype": "aut",
                "href": "c2a0e6be-ef72-3db0-bc96-90d0b74f5c07"
            },
            {
                "type": "P2010",
                "subtype": "aut",
                "href": "4bcd7967-25b4-3593-bfb1-ff76fa117f17"
            },
```

```
            {
                "type": "P2010",
                "subtype": "aut",
                "href": "1b330a56-58c2-33a0-8649-17f1cf099b3a"
            },
            {
                "type": "P2010",
                "subtype": "aut",
                "href": "7829ce9c-c133-3eea-9b9b-2907a1d90019"
            },
            {
                "type": "P2010",
                "subtype": "aut",
                "href": "29066e86-7431-30d7-9875-113d390034c6"
            },
            {
                "type": "P2010",
                "subtype": "aut",
                "href": "6285fb91-a37f-3a2f-9cd5-b8ad5c572d99"
            },
            {
                "type": "P2010",
                "subtype": "aut",
                "href": "e25c5d47-e163-3fdd-a0a3-bad26b363636"
            },
            {
                "type": "P2010",
                "subtype": "aut",
                "href": "6871a154-52b6-3518-a188-aaaf6b7e533a"
            },
            {
                "type": "P2010",
                "subtype": "aut",
                "href": "aef836ec-e285-3776-8d1a-6ad5315b2db8"
            }
        ],
        "label": [
            "Christie, Agatha",
            "Christie, Agatha,"
        ]
```

```
143     },
144     "product": 0,
145     "root": false,
146     "hit": true,
147     "used": false,
148     "children": []
149 }
```

First result in the query “Agatha Christie David Suchet”:

Listing C.2: Query result: “Agatha Christie David Suchet”


```
{
    "id": "3c34d90f-c2ee-3be9-b2b0-2164a5964ce6",
    "name": "3c34d90f-c2ee-3be9-b2b0-2164a5964ce6",
    "type": "person",
    "score": "2.253099",
    "doc": {
        "id": "3c34d90f-c2ee-3be9-b2b0-2164a5964ce6",
        "type": "C1005",
        "datafield": [
            {
                "tag": "700",
                "ind1": "1",
                "ind2": "0",
                "subfield": [
                    {
                        "code": "4",
                        "type": "marc:M700104",
                        "#text": "act"
                    },
                    {
                        "code": "a",
                        "type": "P3039",
                        "#text": "Suchet, David."
                    }
                ]
            },
            {
                "tag": "700",
                "ind1": "1",
                "ind2": "0",
                "subfield": [
                    {
                        "code": "4",
                        "type": "marc:M700104",
                        "#text": "nrt"
                    },
```

```
                        {
                            "code": "a",
                            "type": "P3039",
                            "#text": "Suchet, David."
                        }
                    ]
                }
            ],
            "relationship": [
                {
                    "type": "P2014",
                    "subtype": "nrt",
                    "href": "0a574c2c-c0f6-3beb-a371-21af7b2f47d7"
                },
                {
                    "type": "actorIn",
                    "subtype": "act",
                    "href": "58b70ab8-3a84-3ea8-a597-705e09949545"
                }
            ],
            "label": "Suchet, David."
        },
        "product": 0,
        "root": false,
        "hit": true,
        "used": false,
        "children": []
}
```

Related expression on the search result in apendix C.2.

Listing C.3: Related expression on query “Agatha Christie David Suchet”


```
1
2  {
3    "id": "0a574c2c-c0f6-3beb-a371-21af7b2f47d7",
4    "name": "0a574c2c-c0f6-3beb-a371-21af7b2f47d7",
5    "type": "expression",
6    "score": "1.2762135",
7    "doc": {
8        "id": "0a574c2c-c0f6-3beb-a371-21af7b2f47d7",
9        "type": "C1002",
10       "controlfield": [
11           {
12               "tag": "008",
13               "type": "data",
14               "#text": "100929s2001 caunnn e f eng d"
15           },
16           {
17               "tag": "008",
18               "type": "data",
19               "#text": "070227s2006 caunnnnes f n eng d"
20           }
21       ],
22       "datafield": [
23           {
24               "tag": "041",
25               "ind1": "0",
26               "ind2": " ",
27               "subfield": {
28                   "code": "d",
29                   "type": "P3011",
30                   "#text": "eng"
31               }
32           },
33           {
34               "tag": "240",
35               "ind1": "1",
36               "ind2": "0",
```

```
37                 "subfield": {
38                     "code": "l",
39                     "type": "P3011",
40                     "#text": "eng"
41                 }
42             },
43             {
44                 "tag": "245",
45                 "ind1": "1",
46                 "ind2": "0",
47                 "subfield": {
48                     "code": "a",
49                     "type": "P3008",
50                     "#text": "Murder on the Orient Express"
51                 }
52             },
53             {
54                 "tag": "245",
55                 "ind1": "1",
56                 "ind2": "0",
57                 "subfield": {
58                     "code": "a",
59                     "type": "P3008",
60                     "#text": "Murder on the Orient Express : a Hercule Poiro
61                 }
62             },
63             {
64                 "tag": "336",
65                 "ind1": " ",
66                 "ind2": " ",
67                 "subfield": {
68                     "code": "a",
69                     "type": "P3009",
70                     "#text": "spoken word"
71                 }
72             }
73         ],
74         "relationship": [
75             {
76                 "type": "P2002",
```

```
77                  "href": "1b330a56-58c2-33a0-8649-17f1cf099b3a"
78              },
79              {
80                  "type": "P2003",
81                  "href": "a7393f49-c45b-355c-8ff0-21cd9120ffd0"
82              },
83              {
84                  "type": "P2003",
85                  "href": "2301a480-60d0-35bb-a19e-1d9ca487c67e"
86              },
87              {
88                  "type": "P2013",
89                  "subtype": "nrt",
90                  "href": "3c34d90f-c2ee-3be9-b2b0-2164a5964ce6"
91              }
92          ],
93          "label": "spoken word/eng"
94      },
95      "product": 0,
96      "root": false,
97      "hit": true,
98      "used": false,
99      "children": []
100 }
```