Query Model and Methods for Information Retrieval from Metadata-Based Databases

MATTHIAS KARLSSON

Master of Science Thesis
Stockholm, Sweden 2010
Query Model and Methods for Information Retrieval from Metadata-Based Databases

MATTHIAS KARLSSON

Master’s Thesis in Computer Science (30 ECTS credits)
at the School of Computer Science and Engineering
Royal Institute of Technology year 2010
Supervisor at CSC was Kjell Lindqvist
Examiner was Stefan Arnborg

TRITA-CSC-E 2010:172
ISRN-KTH/CSC/E--10/172--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.kth.se/csc
Abstract

In metadata-based databases, stored information is identified by associated metadata. The more such data that is associated, the better the content can be uniquely described. However, a large number of types of metadata prevent users from specifying appropriate queries. In this thesis, a model for representing the metadata of a system to users that easier facilitates information retrieval is presented, based on taking under consideration specific subsets of metadata and then arranging them in a tree structure with constraints. Using this model, the interface for an existing metadata-based database is implemented.
Referat

Modell och metoder för informationsuthämtning från metadatabaserade databaser

I metadatabaserade databaser identifieras information med hjälp av associerad metadata. Ju mer metadata som används, desto bättre kan information beskrivas unikt. Men fler olika typer av metadata försvårar för användare utformningen av utsökningsfrågor. I denna rapport tas en modell för metadata fram som underlättar informationsuthämtning, byggd på att endast ta hänsyn till specifika delmängder av all den tillgängliga metadatan och sedan arrangera dessa i trädstrukturer med ytterligare restriktioner. Denna modell implementeras sedan mot en befintligt metadatabaserat databas.
# Contents

1 Introduction .................................................. 1
   1.1 Target Audience ........................................ 1
   1.2 Background ............................................. 1
   1.3 Problem ................................................ 2

2 Metadata-based databases ................................. 5
   2.1 Data types .............................................. 5
   2.2 Metadata ............................................... 6
      2.2.1 Relationships ..................................... 6
   2.3 Files .................................................. 7

3 Model ........................................................ 9
   3.1 Metadata Tree ......................................... 10
   3.2 Organization .......................................... 10
   3.3 Example ............................................... 10
   3.4 Alternative usage of labels ............................ 11
   3.5 End notes .............................................. 13
      3.5.1 Metadata node parents ............................ 13
      3.5.2 Number of metadata trees ....................... 14

4 Query methods .............................................. 15
   4.1 Connection between metadata tree and query methods .... 15
   4.2 Expression-based query method .......................... 16
      4.2.1 Validation ....................................... 16
   4.3 Data type-forwarding query method ..................... 18
      4.3.1 Mapping of data type ............................. 18
      4.3.2 Input forwarding .................................. 18
      4.3.3 Expression transformation ......................... 19
      4.3.4 Search workflow .................................. 19
      4.3.5 An example ....................................... 20
   4.4 Facet-based query method ............................... 20
      4.4.1 Interpretation of nodes ........................... 21
      4.4.2 Facet ranges ..................................... 21
Chapter 1

Introduction

1.1 Target Audience

The reader of this document is expected to be familiar with common terms used in
Computer Science, but should otherwise serve to be fairly accessible to any reader
interested in the research topic. Where new terms or terminologies are introduced,
care have been taken to make sure they can be properly understood.

1.2 Background

The research presented in this thesis was conducted at Tokyo Institute of Technol-
yogy, Tokyo, Japan under the supervision of Kjell Lindqvist of Royal Institute of
Technology and Professor Haruo Yokota of Tokyo Institute of Technology.

From September 2005 to August 2006 I was able to participate in a Japanese
exchange program called Young Scientist Exchange Program (YSEP)\(^1\). Each year,
approximately 20 students from various universities around the world are invited
to Tokyo Institute of Technology, where this program is held.

While the program encourages students to also enroll in classes at the university
(of which a few also are required), the main focus of the exchange studies is what
is referred to as Sotsuron, or "graduation (senior) thesis".

This research was conducted as my Sotsuron during these exchange studies, in
the Yokota Laboratory\(^2\), Department of Computer Science at said university, where
my main supervisor was Professor Haruo Yokota.

At the Royal Institute of Technology, this research was conducted under the
supervision of Kjell Lindqvist.

\(^1\)http://www.ryu.titech.ac.jp/~ysep/what.html
\(^2\)http://yokota-www.cs.titech.ac.jp/
1.3 Problem

The research was conducted in the area of metadata-based databases, or "MDB-DBs". In such databases, or rather systems since they typically also employ sophisticated infrastructures in order to be able to serve large amount of contents, stored content categorized by metadata (for more information, see chapter 3).

To give a light introduction to the meaning of metadata of which there will be a more detailed view further on, they can be considered to be tags that can be associated with stored content. Tags, that in the most simple instances have names to make them distinct from other metadata and values that connect them to content.

For a simple example, for systems capable of storing books, it is likely that there in one way or another exists a metadata for authors, namely, a metadata used to store information about the authors for the books that are stored. Using this metadata, providers of content can then easily provide information about a book’s authors upon storing, that then separately from the actual content can be used to later retrieve it.

There are today many implementations of systems using metadata to categorize content and in the research that was conducted the main focus was on what can be considered to be "pure metadata-based databases".

Metadata as a concept strives to encapsulate the practice of storing data about data, meaning a set of data that in one way or another describes another set of data. This is obviously a very broad definition and as such, can be seen employed in a wide array of applications. Though the usage of metadata that was of interest during this research was in the area of querying - using metadata as a tool for finding content - this is by far not the only one.

As previously mentioned, in the area of content-retrieval systems which uses metadata as a tool for querying (or metadata-based databases that they otherwise commonly are known as), in the most simple representation of such system, we have files (or simply content) that are stored and then sets of metadata describing the content in a way that is (in most cases) meaningful for human interaction.

The main focus which my research centered around was retrieval of content from metadata-based information systems. For the remainder of this document, any reference to such system is assumed to be modeled according to the discussion in chapter 2.

Files stored in metadata based information systems can have a vast array of metadata associated with them, where basically the more information the better as it only further makes the categorization more specific. However, users of these systems when searching for content are not necessarily interesting in all these details but only a subset of them.

The problem can therefore be specified as follows. Consider a metadata-based database storing huge amount of data, for which there exist a wide array of metadata used to describe the stored content. The metadata are meant to be interacted with by humans, meaning that they have human-understandable descriptions or names
1.3. PROBLEM

and their associated values also have meanings that are easily understood.

Given this setting, how should a user interact with the metadata in order to query such system for content? This implies two kinds of initial problems.

1. A model. In order to present the vast amount of metadata (most likely) present in these kind of systems, there is a need for constructing a model for representing the metadata. This is a model of the metadata that is supposed to be more accessible to a user.

2. After a suitable model has been determined, it has to be applied to the area of querying. That is, with a model that has been deemed suitable for representing metadata to a user, how can it be used for actual querying.

Both these concerns were addresses during this research and will be given detailed treatments in the following chapters.
Chapter 2

Metadata-based databases

Though the main focus of the research was to bring forward different query mechanisms to an existing metadata-based database, to better describe the approaches, in this chapter there will be an attempt to formalize the assumptions that were made, which are then used to properly introduce the model (see chapter 3).

In this chapter and any following, the word system is often used to reference a given metadata-based database understood from context. In this chapter, it is used to reference an arbitrary fictitious metadata-based system to help bring the discussion together.

Though admittedly the formalization done here was made with the system that was being used in mind, the desire was still that the terminology used would be possible to apply to many other similar systems. Indeed, even the system used did not fit the model from this level and certain alterations to the way different things are represented had to be made.

2.1 Data types

To present the different concepts that are used in order of appearance in this formalization, the concept of data types are introduced. Though one can easily imagine simplified systems which are only capable of (or perhaps more accurately, are only interested in) handling one distinct data type (in which case, most likely a string-based data type), it is assumed or expected that many are present and can be used. Indeed, some of the query methods later introduced are dependent on (or perhaps more kindly, works best with) the availability of different data types. However, no specific restriction is placed on the kind of data types present. That is, they can be as specific as integer and as general as any binary object, though one might question the user-friendliness of having users query systems for content using such data type.

With that said, it is half-expected that the available data types will correspond to other typical data types used in ordinary relational databases such as integer, varchar and date. For the sake of the formalization, it will be assumed that the set of available data types in the system is finite and let $\Delta$ be that set. Furthermore,
for each $\delta \in \Delta$, let $D(\delta)$ correspond to the domain of all acceptable values for $\delta$.

As an example, the domain for a data type unsigned integer, $D(\text{unsigned integer})$ would most likely contain all the values in the range $[0, 2^{32})$, assuming a 32-bit architecture.

A thing to note here though before moving on is that while a system might implement further generalized (data) types that can be used, such as for example a type name name used to store various kinds of names (which might for instance impose restrictions on capitalization), it is assumed that these kinds of types in the end are backed by one of the previously mentioned "primitive" types in $\Delta$. In the previously mentioned example, a proper data type backed by such type could for example be varchar.

### 2.2 Metadata

A rather important section for the formalization given the research area, but nevertheless kept as simple and general as possible.

Metadata, as previously mentioned, are pieces of data attached to other data, meant to in some way describe the data that make it easy to access externally. A clear distinction that can here be made is whether the metadata is meant to be accessed by humans or other computer systems. In the former case, most likely there must exist some kind of identifier that can be easily understood, while in latter one might get away with using for example numerical identifiers commonly understood by all involved systems.

Since the area of interest is that of querying, a task performed by humans, only the first case is really considered but there is no reason for why the latter case could not be formalized in a similar fashion.

With that said, metadata in the system is modeled as a 2-tuple $(n, \delta)$, where $\delta \in \Delta$ and $n$ is a human-readable name of the metadata, meant to give an indication of the type of data that is to be expected. Let the set of all available metadata be $\Gamma$. Furthermore, there do not exist two distinct metadata 2-tuples $(n_0, \delta_0), (n_1, \delta_1) \in \Gamma$ such that $n_0 = n_1$.

That is, the names of the metadata are all unique so that they can be used as an identifier for a given metadata, which is a quite realistic limitation considering the metadata is to be interacted with by humans.

#### 2.2.1 Relationships

There is one more topic that is of concern for the formalization of the metadata, namely that of the relationships between two metadata $(n_0, \delta_0)$ and $(n_1, \delta_1)$. That is, how are they viewed by the system and how this is presented to the user.

In fact, depending on how the system is designed, internally there might be a relationship between the available metadata that furthermore implies the way querying is supposed to be made.
However, for this research no such relationship is assumed, or rather, that the metadata in the system can be accessed and used in an independent way. No doubt this is pretty strong limitation for many systems and depending on by how far a system differ from this requirement, the proposed methods may not work as well as intended. Indeed, even with the system that the methods were implemented for, it was not possible to follow this assumption to the fullest.

The reason for this limitation on the metadata was because of the model for the metadata that is introduced in chapter 3. Since the pre-condition for the research was that there are large amount of metadata that needs to be organized and queried, it felt natural to assume the no prior organization existed. Depending on the system, it very much possible that despite the fact that the stored metadata does not suit this formalization, it is possible to make the model in chapter 3 workable, which in the end is much more important.

Indeed, again, this was required in order to make the implementation work (see chapter 5).

2.3 Files

As in the previous section concerning metadata or more specifically their relationships, the same discussion can be brought to the content, or files stored. In the system, can files be linked together? Are files accessed by group, or individually and so on.

Going along with the same argument as just previously stated, as for the files stored in the system, it is assumed that there is no inherit relation between them, other than the fact that they are stored inside the same system. While there is perfectly acceptable to assume that files can relate to one another with the help of metadata (with the help of for example of some data type that was specifically introduced to be able to reference other files), at the lowest storage level there is no such thing. Or rather, even if there is an inherit organization of the files, it is assumed that they can still be accessed in a flat manner, disregarding any such structure.

Furthermore, as this is a metadata based system, files most likely have a wide array of different metadata associated with them. Adding to the model of the metadata from the previous section, a metadata value (meaning, a value for a specific metadata, tied to stored file) for a file is modeled as \(( n, \delta, v \rangle \), or simply just \(( n, \delta, v \rangle \) where \( v \in D(\delta) \). The set of all metadata for a given file, represented as \( \phi \), is then thereby made up of an arbitrary amount of these metadata values.

Several entries in \( \phi \) for the same metadata \(( n, \delta \rangle \) but with different values is accepted, in order to have content that have several distinct values for a specific kind of metadata information. A paper might for example have many authors.

Again, a limitation that one might find in actual implementations of these kind of systems is those that concern which metadata that are available for use by content. This is also partially connected to the discussion in section 2.2.1, where there in
a particular system certain metadata might only be available once other metadata have been used.
Chapter 3

Model

After having properly formalized the definition of a metadata-based system in chapter 2 it is now possible to present the model for the metadata used for the research and for the proposed and implemented query methods (see chapter 4).

One of the problems that had previously been noted when using the system that the researched query methods should be implemented for was that one often used specific subsets of the available metadata in $\Gamma$. Or rather, as specific type of content was usually queried for, it became quite natural which subset of metadata in $\Gamma$ that were of most importance. However, since this subset was still in most cases considered too big to be accessed in a flat manner (meaning, all metadata there within individually), some other kind of structure was still needed.

That is, it was reasonably to assume that not all of the metadata will be of interest. Furthermore, that it is often possible to identify usage scenarios, such that each scenario has a distinct set of metadata that is utilizes most of the time. Using this observation, by limiting queries to the system into a set of scenarios, the problem is reduced to a subset of metadata that is under concern.

In other words, finding a model for representing an arbitrary subset of $\Gamma$ that simplifies the task of finding content for users. Given that different users have different needs and different kind of content make use of different kind of metadata, the approach is to make a model for task specific subsets of $\Gamma$.

To exemplify the previous paragraph, consider a system for storing digitalized multimedia. We have audio clips and videos, different formats and different contents. The audio clips could be songs or teaching materials, the videos could be movies or documentaries. With the potential for much difference in the content stored, a lot of metadata is created to help the categorization. Different users have different needs for this imaginary system and individual users need different content depending on the situation. The direct approach of giving users the ability to fill out criteria for possibly all metadata in $\Gamma$ would get out of hand as soon as the number of metadata grew, not to mention the fact that it would require users to get accustomed with all the finer detail of the system even for the smallest task.

We now want to have a way for structuring specific subsets of $\Gamma$, each subset
being a scenario. For the remainder of this section, consider $\gamma \subset \Gamma$, an arbitrary subset of $\Gamma$. How should the metadata in $\gamma$ be organized?

3.1 Metadata Tree

In order to find a suitable model for representing subsets of metadata in $\Gamma$ many were evaluated. The need for the model was to be able to represent relationships between metadata in a way that is easily understood, yet reasonably flexible. Because the model would in one way or another (depending on query method) be exposed to a user wanting to query the system, there were natural limitations involved on how complex such model could be.

The two main models that were considered were one where metadata could be related to each other in a graph-like structure and the simpler case where a tree-structure would be used.

Somewhat dependent on the actual system that would be used for the implementation, ultimately it was decided that a tree-structure would be used, dubbed metadata tree, for the representation. The decision for the usage of the structure was partly based on its simplicity and relatively complexity in the sense of being able to express relationships, but also because it is arguable a structure that most people are already reasonably familiar with.

3.2 Organization

To help facilitate such arrangement, a new kind of metadata called label is introduced, represented as 2-tuple $(n, \emptyset)$. Furthermore, for the sake of the tree structure there is also introduced an implied root label represented as $(\emptyset, \emptyset)$ which always exists, unlike regular labels which can be created and/or removed at will.

It is worth noting at this point that a label (including the root) is not part of the actual system, that is $\Gamma$ (and therefore neither $\gamma$), only the metadata tree which is treated as an external entity.

To go further with the concept of a metadata tree, it becomes a necessity to also define the concept of a node in such tree. A node in a metadata tree either represents a metadata in $\gamma$ or a label (including the root). The common element of each node is that they make public a string, representing the name of that particular node. For the sake of generality, this will be assumed to be the name-part in the tuple-representation for both metadata and labels.

3.3 Example

The exemplify the previous discussion, consider a system that among other things store multimedia content such as books, movies and songs. For this reason a wide array of different metadata meant for such content has been created. Users now want
to retrieve such content. For this scenario, after looking at the available metadata it has been deemed suitable to expose those in table 3.1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artist</td>
<td>Name of artist for track</td>
</tr>
<tr>
<td>BookAuthor</td>
<td>Author of book</td>
</tr>
<tr>
<td>BookTitle</td>
<td>Title of book</td>
</tr>
<tr>
<td>BookYear</td>
<td>Year book was released</td>
</tr>
<tr>
<td>ContentSize</td>
<td>Size of content</td>
</tr>
<tr>
<td>Director</td>
<td>Director of movie</td>
</tr>
<tr>
<td>MovieName</td>
<td>Year movie was released</td>
</tr>
<tr>
<td>MovieYear</td>
<td>Name of movie</td>
</tr>
<tr>
<td>TrackName</td>
<td>Name of track</td>
</tr>
<tr>
<td>TrackYear</td>
<td>Year track was released</td>
</tr>
</tbody>
</table>

Table 3.1. An example of metadata subset

As was explained, with a metadata tree we will give a structure when there is none to metadata for specific scenarios, with the use of labels. For the example given in table 3.1, considering it contains metadata for content based books, movies and songs and might be natural to introduce labels based on those. That is, introduce the three labels \((Books, \emptyset)\), \((Movies, \emptyset)\) and \((Songs, \emptyset)\).

Using these three labels, a possible metadata tree for this scenario of metadata could be arranged according to figure 3.1.

11

3.4 Alternative usage of labels

First after the model from the previous sections had been accepted and used, an alternative approach to grouping of common metadata was introduced, dubbed grouping labels. Whereas labels as they were introduced in section 3.2 are used to group metadata that (most likely) share a common content type for which they are typically utilized, grouping labels are used to group metadata that have similar meanings, yet not necessarily are as closely tied as those to labels.

To exemplify, consider again the metadata from table 3.1. We have three different metadata related to storing a year, \((BookYear, integer)\), \((MovieYear, integer)\) and \((TrackYear, integer)\) (suitable data types chosen for the sake of this example).

In the same way, a similar relationship can be seen between the metadata \((BookTitle, varchar)\), \((MovieName, varchar)\) and \((TrackName, varchar)\), and to some extent, \((Artist, varchar)\), \((BookAuthor, varchar)\) and \((Director, varchar)\) as well.

The idea for group labels is to be able to take similar metadata like the three examples above and create a label that encapsulates them all.
The requirement for being able to create a group label is that the set of metadata that it encapsulates, or groups, share a common data type in order to eliminate possible validation errors based on input. A suitable name relating the metadata together should then be chosen.

Following these principles, the following three group labels could be created out from the three recognized relationships from before.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Creator, varchar)</td>
<td>(Artist, varchar), (BookAuthor, varchar), (Director, varchar)</td>
</tr>
<tr>
<td>(Name, varchar)</td>
<td>(BookTitle, varchar), (MovieName, varchar), (TrackName, varchar)</td>
</tr>
<tr>
<td>(Year, integer)</td>
<td>(BookYear, integer), (MovieYear, integer), (TrackYear, integer)</td>
</tr>
</tbody>
</table>

Table 3.2. Example of group labels

With these group labels, the tree from figure 3.1 could be simplified to the one in figure 3.2.

From this is quite apparent that group labels and normal labels as they were introduced more or less identical, if it wasn’t for the requirement that metadata in a group label needs to all share the same data type. Though, as seen in figure 3.2 it can be seen that the metadata in each group label is not visible. This is to empathize that since the metadata there within share a common meaning, there is little to no use to make such distinction between, in contrast to the metadata.
Figure 3.2. An example of metadata tree using group labels

positioned under a label.

However, it is worth pointing out that depending on the application, it might be desirable from a user-point-of-view that one is able to determine which metadata that is part of the grouping, even if one is not able to interact with them on an individual bases.

3.5 End notes

As an end note before moving on to the query methods that will make use of the model introduced in this chapter in order to query MDB-DBs for content, this section will be spent on a few lighter topics regarding the metadata tree that did not fit in the previous discussions.

3.5.1 Metadata node parents

The first of these topics is metadata nodes in metadata trees, that is, nodes in a metadata tree that references a metadata, and whether they should be able to act as parent nodes.

During the initial development of the model and query methods to come, this was not considered as no obvious semantic meaning of such arrangement could be found. However, as it turns out, for a particular method introduced in section 4.4, allowing metadata to be arranged as such could in some cases be very helpful.

In order for the semantic to work however, it is required that there exist some kind of relationship between values of different metadata, namely those kind of relationships described by functional dependencies[2].

This will be further detailed in the relevant coming sections, but for now it is sufficient to state that it can be desirable to group metadata in such why that their associated values are accessed in the same way functional dependencies explain relationships.

As a short example, consider the simple functional dependency

\[ X \rightarrow Y \]
where both \( X \) and \( Y \) are single-values attributes, both referencing their own metadata. If there would exist a relationship between the associated values for the metadata \( X \) and \( Y \) in the way the given functional dependency describes, then, with for example the method in section 4.4 it would be desirable to able to have \( Y \) be a child of \( X \) in the metadata tree.

The drawback being though if one would go about modeling a metadata tree based on the dependencies among the values of the metadata is that the relationships can only be expressed naturally in the cases when the left-hand side of the function dependency is single-valued. Though multi-valued left-hand sides can be implemented in the form of additional tree-levels, it is questionable if such arrangement is one of practical nature.

### 3.5.2 Number of metadata trees

The second of these end notes concerns the number of metadata trees that are present.

As previously mentioned, the purpose of a metadata tree was to group metadata related to a common task or user scenario. That is, take subset of the metadata in \( \Gamma \) create a relationship between them in a tree-structure, possibly making use of the also previously introduced labels or group labels.

It is therefore reasonable to assume that in a practical MDB-DB with a \( \Gamma \) that have far more metadata than what is practical to have in but one metadata tree, that there will be several. With this in mind, it is probably desirable that one is able to associate a human-understandable identifier to each metadata tree so that they more easily can be referenced to, when their usage are needed.

While this is partially an implementation issue, it can be left noted here that it is expected that there for each metadata tree at the very least is associated a name describing its purpose.
Chapter 4

Query methods

The term *query method* refers to a method used to query MDB-DBs for content. After having decided on a generic model suitable for representing metadata that was deemed superior to the unstructured "flat" approach, it was to be applied in the area of querying.

For this reason, several query methods were proposed that made use of the model introduced in chapter 3 and in this chapter, three of those are presented.

Then later, in section 4.5, their different merits are compared in a brief comparison using the implementation that was conducted (for more information, see chapter 5.

The names for the query methods are based loosely on the way they work.

### 4.1 Connection between metadata tree and query methods

Before actually introducing the query methods, this part will attempt to clarify the connection between the model, that is, the metadata tree from section 3.1 and the query methods that are about to come.

In layman’s terms, the metadata tree can be seen as a nail and the query method a hammer. The material that the nail is to be hammered into is the content that are to be retrieved. In order to find the content that one desires, one first picks a suitable metadata tree (which nail goes best with what material?), and then a query method that best fits the metadata tree for the current query (which hammer is most suitable?).

That is, the idea being that when querying a MDB-DB making use the metadata tree and the coming query methods, a user would first pick a metadata tree containing the metadata that is most likely to suit the needs for the content that are to be retrieved. Here there is a slight connection to the end note discussions from section 3.5.2, concerning the desire that metadata trees should have a name (and possibly a description).

It was also mentioned above that one after having selected a metadata tree,
would also pick which query method to use. Going slightly ahead of the conclusion that is drawn in section 4.5, what this means is that it was never a desire to come up with the one query method that would be used in all cases. Rather, recognize that different look-up mechanisms are desirable in different situations and use this fact to allow a user to have a choice.

The model, in this case the metadata tree, is however expected to be fixed as the relationship between metadata is thought of to be independent on the way querying is made and the idea being to introduce query methods that make use of these relationships in new ways. This is similar to the characteristics of the architectural pattern Model-view-controller, originally introduced in [8].

With the metadata tree being the model, the query method would function as a kind of combination of controller and view, controlling how the metadata is handled and how the metadata tree should be presented to the user.

4.2 Expression-based query method

Expression-based query method was the first query method that was brought forward and implemented.

As a quick start note before moving forward, even though, as is laid out in section 4.5, it in the end turned out to be too complicated for normal usage scenarios (that is, it failed to take a big step away from the complexity that a flat representation imposed), for completeness sake, it is presented here.

The goal, or motivation for this method was to see the metadata tree as a tool for a task that was to be conducted. Meaning, that the metadata present in a metadata tree represented information that needed to be filled out in order to conduct the task that the metadata tree itself was responsible for.

Visually, the user a given a clear view of the entire metadata tree, with input fields available for each node, include labels and group labels.

Seeing as a given metadata tree contains metadata for specific content from the connected MDB-DB, this method would take this one step further and control how input should be entered to archive this goal. Or rather, which input that should be entered and in which order.

This query method can therefore be seen as a textual query method, meaning, it is controlled and used by textual input. The query method in section 4.4 takes an alternative approach.

The two concepts used for accomplishing this are called Validation and Limitation.

4.2.1 Validation

The first one of the two concepts, Validation is not strictly part of the goal or work-flow that this query method attempted to accomplish, but since it is tightly connected to the concept of textual input it is presented here within.
4.2. EXPRESSION-BASED QUERY METHOD

It would be preferable if the method was in some respect self-validating. In the simplest instances, input should be validated according to the data type of the metadata it belonged to. Only alphanumerical entries should be allowed for integer values and so on. This validation should also, where possible, be performable as close to the interaction, or view, as possible.

To model this, a mapping function

\[ C: (v, \delta) \rightarrow \text{bool} \]

is introduced. This validation function is used to verify the correctness of the input for a specific metadata. That is, given a value \( \{n, \delta\}, v \), this function maps to true if and only if \( v \) is valid in respect to the data type \( \delta \).

\[
\begin{align*}
C(v, \delta) : \\
\text{if } v \in D(\delta) \text{ then TRUE} \\
\text{otherwise FALSE}
\end{align*}
\]

\( C \), by itself is however not very interesting and it is to be expected that the metadata system in use already has some mechanism for this already in place. If the language used to interface with the system during queries is for example strongly typed, the API probably already enforces this by itself. The function will however be used in subsequent discussions.

\( C \) is here a decent solution to the problem about input being verified as correct or not. The one remaining is about what should be filled in. In this method’s current state, there is no mechanism for stating what should be filled in or in what order. That is, even if there is now a visual structure to the metadata, there is no such thing as for the order of input.

To exemplify, consider a locked down version of figure 3.1 where one can only search for either book, movie or song content, but not any of them together. This might be to simplify the search process or because of limitations in the underlaying system. Nevertheless, the goal is to be able to have criterials for when metadata is available for entry, preferably combined with the tree structure that already is in place.

The proposed solution is a combination of boolean logic and the \( C \)-function that was previously introduced. The idea being that one for each node in the tree should be able to construct a boolean expression for under which conditions it is valid. For this purpose, an additional mapping function

\[ V: (n, \delta), v \rightarrow \text{bool} \]

is introduced that given a metadata and its associated input value, is true if and only if it satisfies its boolean validation expression. The boolean validation expression should be a boolean expression formed with its (if any) children nodes.
4.3 Data type-forwarding query method

Inspired by the expression-based query method (see 4.2), this query method became the natural successor, much due to its simpler approach and perhaps more importantly, simpler use.

By using the data types that are associated with each metadata, the idea is to identify data types from input and then perform input forwarding to appropriate metadata. With this method, when presented with a metadata tree, it not only accepts input from nodes associated with metadata, but labels as well, including the root. Input from the labels are then pre-processed based on data type and forwarded to appropriate child nodes, based on a pre-determined mapping.

4.3.1 Mapping of data type

For each label (including the root) one can set up mappings from data type to child nodes. That is, for each label \((n, \emptyset)\), for each \(\delta \subseteq \Delta\), it is possible to associate a list of child nodes to which input is to be forwarded. Though one can here imagine the mapping being dynamic, meaning that the data type mapping is made automatically based on child nodes, in order to be able to prioritize different metadata over others, it will be assumed to be a manual process.

Since labels can be child nodes of other labels (most commonly the root label), by setting up mappings that forward to a child label node that in turn has its own mappings, it is possible to get input forwarded recursively down a metadata tree.

4.3.2 Input forwarding

When a user is presented with a metadata tree using this method, the idea is to only show one input field, the one associated with the root label. In this field, the user is assumed to enter information that is known about the content that is to be received, just like one would typically use an every-day search engine.

When a query is submitted, input from the root label is received and processed. This is done by tokenizing the input into a set of tokens \(\Sigma\). Each \(\sigma \subseteq \Sigma\) is then checked against each \(\delta \subseteq \Delta\) whether \(\sigma \subseteq D(\delta)\) or not. If yes, the mapped child nodes for that data type is looked up for the root node, and \(\sigma\) is forwarded to its input.

After the processing of the input from root label has been completed, the metadata tree is recursively checked for more labels and for each one found, the forwarding process is repeated.

A point of discussion here is whether forwarding should be made for all matching data types or only the first match. In the implementation that was done, the data types of \(\Delta\) are checked in a pre-determined order and the forwarding is halted whenever a match is found.
4.3.3 Expression transformation

To prepare the metadata tree with processed input for a query that the underlying MDB-DB can understand, a boolean expression is formed. Just based on the processing that has been done according to the discussion above, each nodes associated with the metadata tree is processed and for each token in its input, an boolean expression is formed. All expressions are then gathered and assembled into a final expression with the operand OR.

For this reason, a query entered into the root label and submitted will potentially be subjected to a very broad search and the tokens can be forwarded to a wide array of different metadata that may or may not be of the kind that was thought of. To remedy this, a user can in some way in the user-interface tell the root label to expand to show its child nodes. This operation can be done for any label. The user will then get a clearer picture of how the input was mapped to different metadata. By looking at the metadata now seen, the user is now able to tell the user-interface that input forwarded to a specific node is a strong match, called determining a node.

This has two effects. For one, it tells that the input for this particular node should be considered necessary in order for content to be matched and it also stops manipulation of its input based on input forwarding, should its parent label input be altered. In case a label node is determined, it will stop further forwarding to that node and in case of a metadata node, the way the query expression is formed is altered.

The more generalized way the query expression is formed can now be defined. After having performed input forwarding to nodes that has not been marked as determined, all nodes associated with metadata are processed. For each determined node, an expression is formed. The expressions from the determined nodes are then combined with the operand AND. Finally, an expression is formed according to the discussion before and also added with the operand AND to form an expression.

For each node, there is also possible in the interface to mark whether separate tokens must all be valid, or if only one of them. That is, when an expression is formed for that particular input, if the individual expressions should be combined with AND or OR. Default is OR.

4.3.4 Search workflow

The workflow that is expected is that user first enters known information into the only shown text input field and after submission, it is parsed and forwarded to appropriate child nodes and content retrieval is performed based the query expression that is built. While the goal is that this should be the only necessary step that the user has to take, depending on the metadata in the metadata tree and the input from the user, many unrelated content material may have been found. A user then expands the root label and determines any nodes to which input has been properly forwarded, each time narrowing down the number of hits.

While nodes associated with metadata are also subject to user input and the
user can also edit their content, this is not the original idea. In fact, it is very much possible to bypass the entire concept of forwarding by simply expanding the root label and then filling in (and determining) the metadata that are of interest. For an experienced user, this might even be desired if one is familiar with the structure of the metadata tree.

For the way this method was meant to be used, it is most suitable for MDB-DBs that do not contain a lot of similar information between different metadata. Should that be the case, after a first input to the root label when a very broad search is performed due to the data type forwarding and no determinations, the likelihood of many unrelated hits can be high. This can be countered by later determining proper nodes, but it is also possible to limit the impact by fine-tuning where forwarding should be made, depending on data type.

4.3.5 An example

To give a descriptive example, consider figure 3.2 as if it would be used with this method. At first, the user would only be presented with a text field associated with the root node. In here, information for the kind of media content that is of interest would be entered. For finding a specific song, one might enter the name of the artist, some words from the title of the song as well as a year range for wherein it was thought of to have been released. For this example, it is assumed that mappings have been set up from all labels to all child nodes based on suitable data types as deduced from 3.2.

The name of the artist and the words from the song would be forwarded to \textit{Creator} and both labels and from there recursively to \textit{Album}, \textit{Artist}, \textit{TrackName}, \textit{Country}, \textit{Director} and \textit{MovieName}. The year range would go to the labels and from there down to \textit{TrackYear} and \textit{MovieYear}. Depending now on the amount of content in the MDB-DB and the possible false hits introduced by mappings of input to fields that are unrelated, the user may now receive a good list of possible files. If many files were received, the user can now expand the root label to reveal the mapping of the input and then determine those that are deemed good to further classify the query and hopefully reduce the set of returned files to a more manageable one.

4.4 Facet-based query method

This section will describe a facet-based query method, using as the previous ones a metadata tree. Created as a kind of complement to the previous textual-based ones, this method works primarily on metadata which share common values between files. By using the tree structure given from the metadata tree, it can immediately be used for facet-based querying.
4.4 FACET-BASED QUERY METHOD

4.4.1 Interpretation of nodes
With this method the root label is ignored as it serves no direct purpose other than making the tree structure valid. For nodes associated with metadata, the MDB-DB is queried for a list of unique values. Label nodes serve as containers for related metadata.

4.4.2 Facet ranges
For metadata with a large amount of unique values, it may be necessary to have some kind of limitation to not overwhelm a user with too much information.

For integer-based data types such as Integer and Date, facet ranges can naturally be created based on a min-max-paradigm. For such data types one can consider two different usages of facet ranges; either facet ranges that are created dynamically based on the unique values returned from the MDB-DB, or from a manually constructed set associated with the metadata node. In the former case, based on a threshold, facet ranges could be constructed for a set of unique values of an integer-based type if the number of returned values turns out to be too big.

For other data types such as string-based ones, a natural facet range arrangement could be to categorize according to the initial character.

4.4.3 Search workflow
The idea is that the user is given a view of the metadata tree as it had been expanded by the root label, though itself not visible. Other labels are closed. For each node associated with metadata, a list of unique values for all files using that particular metadata is shown, allowing users to select one. When a value for a metadata is selected, it is removed from the metadata tree and set aside in a visible manner so that users can see the conditions that has been created.

Selecting a facet range value for a metadata creates a condition based on that specific facet range, but the metadata should not be removed from the metadata tree. Instead it should remain with a new set of unique values to select based on existing conditions, perhaps again subject to range limitations (in case of dynamic creation).

By selecting label nodes in the interface, users can expand them to reveal additional metadata and their values that might be of interest. However, not only label nodes but also nodes associated with metadata can act as parents. In the latter case, the requirement for expanding such nodes is that a value is chosen. This allows one to hide information related information between common values for other metadata. A well-defined case is geographical information. Consider a MDB-DB with the metadata Country, Region and City. While they are independent (or as assumed, can be considered being), a valid assumption would probably be that content with a common value for the metadata for City would also share a common value for the metadata for Region. To reduce the number of unique values that are shown to users it may therefore be desirable to position these metadata in a parent-
child-relationship in the tree-structure with City as a leaf-node having Region as its parent which in turn being a child of Country.

4.4.4 Expression transformation

In essence, this method allows user to create necessary conditions on metadata that content must fulfill in order to be matched. For each chosen condition, an expression is formed and they are then combined with AND to form the final query. This is similar to the previous method if one just considers the metadata that was determined. Furthermore, the unique list of values are retrieved for metadata is also always under the constraint of the conditions that has been placed before. This means that one can never reach a situation where no content is found as one always start from the set of all content in the MDB-DB and then narrows it down until a suitable subset has been formed. For proper user feedback it is also expected that the MDB-DB can display the number of hits for each unique value, should one select it.

4.4.5 An example

To, like with the previous method method, give a descriptive example, consider figure 3.1. It is assumed that dynamic ranges have been created for data types with a sensible threshold. Upon first getting a view for it, it would be shown as if the root label had been expanded and for example TrackYear integer ranges would be visible while for Artist there might be a complete list of all names or some kind of groupings based on the initial character. By clicking on Songs the user could reveal the metadata that are of interest and the correct artist could here be chosen and a new filtered view of content matching the selected condition would be shown. It might be worth noting that for this method, a metadata node for track names might not be desirable as it could be expected that it would for the most part contain unique values. However, by filtering the returned content by TrackYear the user is most likely able to get a good filtering or in case the metadata for track name is available (as in the figure), by knowing its initial spelling it could also have been selected.

4.5 Comparison

While previous sections have already contained lighter discussion on respective query methods’ pros and cons, this section aims to bring these together to give a summarized comparison.

With the expression-based query method (section 4.2), it was quickly discovered that while it was perfectly possible to technically use is to retrieve any file, its usage would be too hard to explain to make it usable. The biggest obstacle that led to the improved Data type-forwarding query method (section 4.3.2) was that it was apparent that it was not intuitive how the different expressions should be presented
4.5. COMPARISON

to the user. Furthermore, the fact that the whole metadata tree was to be visible at all times would not be feasible if the metadata trees were to grow to greater sizes than those in the given examples.

Fortunately, the Data type-forwarding query method (section 4.3.2) proved to be much more usable in this regard due to its goal to show as little metadata as possible; in fact, it is possible to use it without seeing any explicit metadata at all. In some way, this query method can simplified to be seen as the most obvious query method where you simply take user input and try to apply it to all the metadata that you have. This can be a problem if the number of metadata is great, but fortunately this problem is implicitly bypassed due to the usage of metadata trees. The main problem with this query method is when there is much overlap in the domains of different metadata belonging to a metadata tree. When this is the case, there is a greater chance for hits from various metadata nodes which in turn increases the risk for false positives. This can be fine-tuned by more carefully constructed metadata trees and forwarding, but it is possible that in practice, other considerations need to be made.

Developed last, the Facet-based query method (section 4.4) was brought forward to complement the previous ones with a method that was not dependent on textual input but one which would instead allow a user to navigate the files of a system using a metadata tree. By selecting specific values for the metadata in the metadata tree that are known to be valid, either explicit ones or through ranges, the files of a metadata system can easily be navigated. Furthermore, reducing result sets is easily done by picking additional metadata from the metadata tree. Using this, a user can hopefully get a clearer picture of the files that are available. However, if most metadata values are unique, it can be hard to pick values for different metadata unless ranging is used in a good way.
Chapter 5

Implementation

This section aims to provide an overview of the implementation that was constructed as part of the research.

By reading this section, it is the intention that one afterwards has a clear understanding of how the implementation works and how the different components play together.

5.1 KnowledgeStore

To create an implementation of the research conducted, access was given to a MDB-DB called KnowledgeStore[13] available at Tokyo Institute of Technology. This implementation of a MDB-DB ended up matching the formalization made in chapter 2, though a few alterations were needed.

While the metadata available had no inherit relations between themselves, they were still categorized into a small set of distinct parts depending on content. KnowledgeStore stored, among other things, large amount of research papers and digitized media. When querying the system one first had to choose which part that was to be queried which further implied which metadata that was available. As such, construction of metadata trees had to be limited to metadata from one such part of the system. While one could easily imagine an implementation that combined metadata from different parts when constructing metadata trees, due to performance reasons this was not a practical approach.

See figure 5.1 for a simplified view of the architecture of KnowledgeStore.

The implementation was constructed as an external system as the figure illustrates which makes it independent from the core parts, accessing its information through a well-defined web service. Through this web service, it was possible to query the system for its sets of available metadata and to query for files using simple boolean expressions.
5.2 Architecture

5.2.1 Overview

To implement an external system that could interface with KnowledgeStore (section 5.1) and at the same time easily allow other users to use it, it was decided that the best approach was to create a web-based platform.

As such, the implementation was done as a web application and was independently hosted. For performance reason, it used a local relational database to store information about all available metadata in the system. This database was also used to store any persistent information needed to implement the query methods that interfaced with KnowledgeStore. Specifically, it allowed the persistence of metadata trees.

The set of metadata from KnowledgeStore was periodically synchronized with the implementation to always allow its users complete access. As the set of metadata in KnowledgeStore was relatively static and therefore rarely changed, this proved to work very well. Had this not been the case, a more sophisticated approach would had been needed.
5.2. ARCHITECTURE

5.2.2 Programming language

The web application was implemented using the Java programming language as a J2EE web application. As a common API in the language had previously being written to interface with KnowledgeStore, this was an obvious decision.

To ease the work needed for the implementation, the Spring Framework\(^1\) was used to separate the code with a Model-View-Controller-pattern, originally introduced in [8]. This allowed for quick development of the implementation’s core parts and made it easy to add or remove parts as the development progressed.

Using Java Server Pages\(^2\), the View was implemented.

5.2.3 Application server

The Apache Tomcat application server\(^3\) was used to host the web application after it had been implemented.

5.2.4 Database

Due to already being available on the host where the implementation was be available, MySQL\(^4\) was used to implement the local storage mechanism. As previously mentioned, it was to persist the metadata trees and to also work as a metadata cache for the metadata available in KnowledgeStore.

\(^1\)http://www.springframework.org/
\(^2\)http://java.sun.com/products/jsp/
\(^3\)http://tomcat.apache.org/
\(^4\)http://www.mysql.com/
Chapter 6

Related works

The definitions that we established in chapter 2 in reference to MDB-DBs are certainly not the first ones made and can be considered being a simplified view. More comprehensive ones have been made, such as in [3], [7] and [10].

The basic motivation for introducing the Metadata tree forwarding method in section 4.3.2 is similar to the four-phase search framework introduced in [9] which concerned interaction with text-based search interfaces. Many of the concepts therein could serve well as guidelines for real-usage implementations of the methods from this paper.

Though the type of content is not implied with neither one of the proposed methods, uses of the Metadata tree-based facet method are very common in the realm of media, especially images and there are many related studies in this area. Examples of such can be found in [12] and [4]. In these researches many similar conclusions are made regarding the ease of use in respect to metadata and how metadata can be used in the field of querying. They also hint on the fact that users recognize the utility of switching to the interface that best supports the current task, while not directly supporting this in their current implementation. This was the main reason for introducing metadata trees, which could then independently be used together with a query method to query for content.

In [11] a comprehensive summary of content-based image retrieval can be found. Furthermore, this area in particular is also subject to many usability concerns, such as mapped out in [5]. Again, which could serve a guidelines for real-usage implementations.
Chapter 7

Conclusion

In this thesis the concept of a metadata tree based on a general definition of metadata and metadata based databases was established. For the purpose of creating such trees we further introduced the concept of metadata labels which are used help the facilitation of metadata arrangements. By identifying common user queries, or scenarios, the metadata tree is used to take subsets of metadata from the MDB-DB and making logical arrangements.

This metadata tree is then used to create specific query methods, one based on textual input named Data type-forwarding query method and one based on facets called Facet-based query method.

In the former one conceptually uses the labels of the metadata tree (including the root) to enter the known information of the content that one wants to retrieve and then based on data type mappings the entered data is forwarded to appropriate child nodes. One can then navigate the metadata tree and assert that any forward that had been made is to be determined, meaning that the mapping was correct in terms of input and the kind of metadata.

The latter one allows a user, based on the structure of the metadata tree, to navigate the content of the MDB-DB by imposing strong conditions by selecting from lists of values that metadata have associated with them. By limiting the amount of information that is presented to the user in cases when the MDB-DB has a vast amount unique values for specific metadata, the concept of ranges is introduced.

The metadata tree and the two methods were then implemented for an already existing MDB-DB called KnowledgeStore [13, 6].
Chapter 8

Further work

After finishing the research for my *Sotsuron* at the Yokota Laboratory at Tokyo Institute of Technology, the foundation was used by following graduate students with the aim to finish a production ready implementation to be used by everyday users of KnowledgeStore[13, 6].
Chapter 9

Publication

Portions of this thesis was additionally submitted to and presented at DBWS2006\textsuperscript{1} in July 2006, under the title \textit{Query model using tree-structure of metadata subsets for metadata-based databases}, authored by Matthias Karlsson, Takashi Kobayashi and Haruo Yokota, under the supervision of Haruo Yokota.

\textsuperscript{1}http://www.ieice.org/~de/DBWS/dbws2006/
Bibliography


