METABASE
A DISTRIBUTED METADATA DATABASES WITH OSF/DCE UUIDS

By
Yangkun Zhang

A Master’s paper submitted to the faculty
of the School of Information and Library Science
of the University of North Carolina at Chapel Hill
in partial fulfillment of the requirements
for the degree of Masters of Science in
Information Science

July, 1999

________________________________
Adviser

Metabase is a model for a metadata database that is based on a relational instead of a flat model. At the core of the Metabase concept is that every document, idea, and concept represented in the metadata repository is represented by an Universally Unique ID (UUID) that is always identical and unique. Hence, every concept can be searched for and replicated across various machines without needing to worry about name collisions. The relational tables in Metabase also allows complex metadata which cannot easily fit into a conventional “flat” metadata scheme to be used.

Headings:

Metadata

Relational Database

Metadata Replication
Overview

With the explosion of the Internet in the past few years, there has been a pressing need for a standardized metadata system to keep track of electronic content, be they multimedia, text documents, or binaries. While there have been many variations of metadata schemes, they all typically flat files and are usually limited to a certain type of data which does not extend well to complex concepts such as multiple files formats of multiple language versions of the same document.

Let's use as an example the English and Spanish version of the NATO charter. If you store metadata information about these documents in your typical metadata database, you will end up with some very interesting search problems. If you search for the topic NATO, you would only come up with the English version of the NATO charter. If you search for OTAN, you would come up with only the Spanish version. A way to resolve this, of course, is reference both NATO and OTAN to both documents. Of course, this is an extremely cumbersome way to go about it--think of what would occur if we had 7 or 8 different language versions of the same document?

What we need is a method to convey in the metadata that both the English and Spanish version of the NATO charter are simply different versions of the same database, and that there is an unique entity that can be described as "the NATO charter". Likewise, the topic "NATO" should be identified as a unique concept, and be declared as an entity that can be searched independent of language specific terms such as NATO or OTAN.

How would be go about this declaration of identity? The simplest way, of course, is to declare an unique ID and assign it to the document/topic--much like social security numbers for Americans. The problem is uniqueness. How can we ensure that an ID created at one place does not "collide" with another? For this we can simply borrow the concept of Universally Unique IDs (UUID) from OSF/DCE 1.1. (See http://www.bu.edu/~jrd/FreeDCE/dce11rpc.html for a free DCE RPC toolkit.) A UUID is a 128 bit "statistically unique" ID generated by a combination of your 48 bit MAC address, the current time, and a large random value. With a unique identifier, we resolve any potential ID collision problems.

Therefore, we simply assign an UUID to any document in our metadata database, so that a document is a unique entity. With UUIDs attached to documents, we can further create unique metadata tags to identify each unique version of the document (e.g., English and Spanish). Furthermore, we can attach multiple physical files to each version of each document (e.g., text/html, application/ms-word, etc.). Every entity in our metadata database shall be uniquely identified.

What does this buy us? Well, if are going to replicate this metadata database over several sites via a publisher/subscriber paradigm, the subscriber can simply check for the existence of a document by checking its UUID. The document's descriptive name can
indeed be totally different, but the metadata repository will, based on UUIDs, know
whether or not two documents are the same.

Furthermore, we can generalize this tagging of unique metadata with UUIDs further by
allowing the metadata implementer to tag *any* arbitrary concept with an UUID.
Document categories can be tagged with UUIDs—thus the same category could be found
via a UUID no matter what language the category description is in. (Think of this as a
dewey decimal system for metadata repositories—the UUID, and likewise dewey
decimal, should remain the same regardless of the descriptive title of that category.)

In a publisher/subscriber replication scheme, the subscriber can get a list of the newly
added UUIDs from the publisher, and simply add them to its own metadata repository
(this is possible as UUIDs are unique and no ID collision is possible). The subscriber can
then replicate the actual data attached to the UUID, be it a descriptive text field or a file.
The subscribers can contribute to the publisher by simply adding a new UUID for this
new document (plus various version information) to its own metadata repository and tell
the publisher, along with the suggested category UUID for this new document. The
publisher can then either add this new document, or ignore it.

Implicit in the publisher/subscriber scheme is that there exist a central authority that
guarantees the goodness of the information in the metadata repository. That someone
actually manually maintains the repository and blesses additional documents as they enter
the corpus of document metadata. This repository is not ideal for an ad hoc "everyone can
contribute freely" collection of documents—in order for this type of repository to
function correctly, there must exist an individual (or a group of individuals) who
"moderate" the contributions from the community at large.

**A Relational Instead of a Flat Metadata Model**

Metabase’s fundamental data model is modeled after a highly modified Dublin Core
Element Set¹. The data model is done using an entity-relational schema as opposed to a
flat one (please see Figure 1 for details of a sample data model).

Of the Dublin Core elements, title, subject, description, and publisher remains the same,
although given that ours is a relational model, we can support many to many relationships
between the core document and publishers and/or subjects. The author/creator and
contributors of the Dublin Core has been flattened into a single attribute—this is because
our relational model allows many to many relationships. Date is attached not to the
document itself but *a particular copy* of the document, as we will see later. Resource type
and format has been replaced with MIME types. Resource identifiers are replaced with
UUIDs. The language attribute exists on a per document basis. The relation is not needed
as relationship is indicated by the use of foreign keys amongst different tables. Finally,
source and coverage is simply ignored. However, additional attributes can be easily

¹ [http://purl.oclc.org/dc/](http://purl.oclc.org/dc/)
added with the addition of extra tables using document UUIDs as the foreign key. Also included in the metadata database is the creation date of each entry—this is quite different from the concept of Date from the Dublin Core, as the creation date refers to the date and time the metadata was introduced to the entire domain of various distributed metadata databases. The attribute "creation_date" is used by the repository for "changes since x date" replications.

The following is a sample entity-relationship diagram of the Metabase repository. Please note that on each entity, there exist an attribute "creation_date"—date on which a particular item was incorporated into the repository—which is not displayed.
Figure 1—the metadata database ER diagram.

The diagram is draw using the IDEF1X (Integration DEFinition for Information Modeling) notation for logical modeling. In IDEF1X, the hallow diamond represents zero or more; the solid circle represents many; the solid line signify an identifying relationship, and the dashed line represents a non-identifying relationship.
Physical Representation Within a DBMS

The following is an example physical implementation (for Sybase and Microsoft SQL Server) of the logical model. Please note that this does not have to be the implementation—the actual database does not even have to be relational (though it does make it easier), it could very well be an OODMBS. Also keep in mind since tables can be more or less added at will, the final metadata repository may not even look like this. It is only required that both the publishers and subscribers of a particular channel of metadata use the same data model.

```
CREATE TABLE Categories (  
cat_id               binary(16),
cat_desc             char(255),
parent_cat_id        binary(16) NULL,
creation_date        datetime,
PRIMARY KEY ( cat_id),
FOREIGN KEY ( parent_cat_id)  
REFERENCES Categories
)  
go
```

The table "Categories" defines just that—a arbitrary category of documents that can have one parent category. If the attribute "parent_cat_id" is NULL, then that category is assume to reside at the root. This allows a tree structure of categories, much like a file system.

```
CREATE TABLE Document (  
doc_id               binary(16),
doc_title            char(255),
doc_desc             text NOT NULL,
creation_date        datetime,
PRIMARY KEY ( doc_id)
)  
go
```

```
CREATE TABLE Document_Categories (  
doc_id               binary(16),
cat_id               binary(16),
creation_date        datetime,
PRIMARY KEY ( doc_id, cat_id),
FOREIGN KEY ( cat_id)  
REFERENCES Categories,
FOREIGN KEY ( doc_id)  
REFERENCES Document
)  
go
```

"Document_Categories" defines the relationship between documents and categories. This is because any particular document can reside in multiple categories.

```
CREATE TABLE Rights (  
rights_id            binary(16),
)  
go
```
"Rights" should contain the full licensing and copyright information.

CREATE TABLE Reviewer (
  reviewer_id          binary(16),
  name                 char(255),
  reviewer_desc        char(255),
  reviewer_email       char(255),
  creation_date        datetime,
  PRIMARY KEY (reviewer_id)
)

The reviewer is the person who has reviewed the content of a particular incarnation of a particular document (referred in the system as a "Doc_Copy", or a copy of a document) and has "blessed" that copy as being worthy of adding to the collection.

CREATE TABLE Doc_Copy (
  copy_id              binary(16),
  doc_id               binary(16),
  rights_id            binary(16),
  language             char(255) NULL,
  reviewer_id          binary(16),
  country              char(3) NULL,
  format_mime          char(255),
  review_date          datetime NULL,
  url                  text NULL,
  creation_date        datetime,
  PRIMARY KEY (copy_id),
  FOREIGN KEY (rights_id) REFERENCES Rights,
  FOREIGN KEY (doc_id) REFERENCES Document,
  FOREIGN KEY (reviewer_id) REFERENCES Reviewer
)

"Doc_Copy" is one of the more important concepts of Metabase. A document in Metabase is simply a concept, the "Doc_Copy" (the copy of the document) is the actual realization of that document. Each "Doc_Copy" can be a different file format of the same document, or it can be the same document translated into a different [human] language. For example, the document "Sonata No. 21 in C, Op. 53" by Beethoven is a concept, or the "Document". A PCM encoded "WAV" audio file of the sonata, or a MPEG-2 video of the performance of the sonata being played by Alfred Brendel, or a MIDI transcription of the sonata, or a scanned image of the score would all be different realizations (or a different "Doc_Copy") of the same document.
The attribute "url" deserves some attention. The "url" attribute describes the URL of the actual document if and only if it resides on a third party server. If the "url" attribute is NULL, the actual document is considered to be residing on the same server as the Metabase, and can be retrieved via a standard method. During replication, the subscriber is free to retrieve the document copy from the source server and store it on its own server, and hence change the "url" attribute to point to its own file repository.

CREATE TABLE Publisher (
  publisher_id         binary(16),
  publisher_desc       char(255),
  creation_date        datetime,
  PRIMARY KEY ( publisher_id)
)
go

CREATE TABLE Document_Publisher (
  doc_id               binary(16),
  publisher_id         binary(16),
  creation_date        datetime,
  PRIMARY KEY ( doc_id, publisher_id),
  FOREIGN KEY ( publisher_id)
    REFERENCES Publisher,
  FOREIGN KEY ( doc_id)
    REFERENCES Document
)
go

As the above indicates, each document can have multiple publishers.

CREATE TABLE Creator (
  creator_id           binary(16),
  creator_name         char(255) NULL,
  creator_desc         char(255),
  creator_email        char(255),
  creator_login        char(255),
  creator_passwd       char(255),
  creation_date        datetime,
  PRIMARY KEY ( creator_id)
)
go

The difference between the "Creator" and the "Publisher" is the same as the difference between an author and publisher.
A subject is not a keyword—it is (like everything else in Metabase) a concept. Using a previous example, NATO and OTAN are different keywords, in English and Spanish, respectively, but they are the same concept, and hence, the same subject, worthy of assigning an UUID to.

```sql
CREATE TABLE Document_Creator (
    doc_id               binary(16),
    creator_id           binary(16),
    creation_date        datetime,
    PRIMARY KEY (doc_id, creator_id),
    FOREIGN KEY (creator_id) REFERENCES Creator,
    FOREIGN KEY (doc_id) REFERENCES Document
) go
```

Each document can have multiple creators. For example, the book *Perl Cookbook* would have two creators: Tom Christiansen and Nathan Torkington. O'Reilly would be the publisher.

```sql
CREATE TABLE Document_Subject (
    doc_id               binary(16),
    subject_id           binary(16),
    creation_date        datetime,
    PRIMARY KEY (doc_id, subject_id),
    FOREIGN KEY (subject_id) REFERENCES Subject,
    FOREIGN KEY (doc_id) REFERENCES Document
) go
```

The UUIDs are represented by as "binary(16)", or an 128 bit binary string. Where DCE UUIDs are presented textually in the format of "7fe2e880-3304-11d3-a167-00a024bafceb", within the Metabase system, they are simply stored "as is" in the network byte order; that is, one can simply strip out the "-" characters, e.g., as "7fe2e880330411d3a16700a024bafceb". In databases that does not have the binary datatype (e.g., PostgreSQL and MySQL), a substitute type such as char(32) may be used—albeit at a slight loss of efficiency.

All date and time information should be translated into GMT.

**XML Representations**

XML is the standard method of representation for metadata and is indeed the ideal way to transport metadata between various machines and applications.
The Dublin Core XML representation is simple enough. Show by the example below, all Dublin Core elements are mapped onto corresponding XML elements as described by the Dublin Core DTD.

```xml
<DC:CREATOR>Rick Jelliffe</DC:CREATOR>
<DC:CONTRIBUTOR xml:lang="zh-TW-Lt">Chin-Tang Chang</DC:CONTRIBUTOR>
<DC:SUBJECT xml:lang="en">XML, SGML, Chinese, FAQ, Big5, GB2312, Unicode, ISO 10646, UTF-8, UTF-16,
Apache, Voyager</DC:SUBJECT>
<DC:PUBLISHER xml:lang="en">Computing Centre, Academia Sinica, Taiwan</DC:PUBLISHER>
<DC:TYPE xml:lang="en">Text.Article</DC:TYPE>
<DC:DATE>1999-04-10</DC:DATE>
```

The Metabase XML representation is just as straightforward, given

```xml
<?xml version="1.0" standalone="yes"?>
<METABASE:Document>
  <METABASE:doc_id>c619b31e-32a6-11d3-8f4c-00a024bafceb</METABASE:doc_id>
  <METABASE:doc_desc>"Waldstein" Sonata</METABASE:doc_desc>
  <METABASE:creation_date>Mon, 05 Jul 1999 07:07:22 GMT</METABASE:creation_date>
</METABASE:Document>
```

the application parsing knows to translate it into the corresponding SQL statement to store the data in its respective DBMS. For example, the above will translate into

```sql
insert into Document(doc_id, doc_title, doc_desc, creation_date)
values(0xc619b31e32a611d38f4c00a024bafceb,
      'Sonata No. 21 in C, Op. 53',
      '"Waldstein" Sonata',
      '05 Jul 1999 07:07:22')
go
```

Of course, the above represents only one of the many tables involved. Other XML elements are similarly defined for other tables.

**Replication of Metadata**

Metadata replication is achieved through a very simple mechanism. It is done through HTTP GET requests. The Metabase HTTP interface can be implemented as a Java Servlet, a CGI, an ISAPI DLL—it doesn't really matter. Each Metabase server is referred to through the full URL of that executable stub, e.g.,
Arguments are passed in via HTTP GET variables. There are only six relevant variables: uuid, table, id_type, xml, begintime, and endtime.

The variable uuid identifies the UUID of the entity you want to return; if this field is left blank, then the UUIDs of all rows of the table (entity) specified will be returned with possible restrictions due to filtering.

The variable table refers to what kind of entity you want to return; this should correspond to the table name, e.g., "Document" for any entity in the "Document" table; this field is also required.

id_type is a required field that work in conjunction with the table field—it describes what types of UUID you would like to search for. For tables such as Document, this is a non-issue, as searching the publisher for information on an UUID will return information on the UUID (which would also be the primary key). However, there are plenty of tables such as Doc_Copy which have two or more attributes which are UUIDs, such as doc_id, copy_id, etc. In this case, an id_type variable is required to distinguish which UUID you would like to search for. Note that multiple results may (and often will) be returned.

The variable xml can either be a 1 or 0, defaulting to 0, and is therefore optional. "xml" describes whether you want the output of the server in XML, or a raw dump of the data in text/plain format in the following manner:

```plaintext
table='Document';
doc_id='c619b31e32a611d38f4c00a024bafceb';
doc_title='Sonata No. 21 in C, Op. 53';
doc_desc='"Waldstein" Sonata';
creation_date='05 Jul 1999 07:07:22';
```

The raw dump format should always start with the line `table='Entity Name'` and lists each attribute in the entity and then the value of that entity. All values should be enclosed in single quotes, with " to denote recurrences of ' within the value—ala SQL. Each attribute value should end with the ';' character.

The fields begintime and endtime are optional. These two variables filters a date and time range for the returned data, and is used when returning only UUIDs of new rows added since a given date.

For replication to occur, the subscriber simply has to connect to the publisher, and ask for all the new UUIDs created in the publisher's repository. For example, the subscriber
wishes to obtain all documents added to the publisher since July 4, 1999. The subscriber would open a connection to the publisher via HTTP and make this request:

```
GET /Metabase.cgi?table=Document&xml=1&begin_time=04%20Jul%201999
```

The publisher would then return the following:

```
<?xml version="1.0" standalone="yes"?>
<METABASE:Document>
  <METABASE:doc_id>c619b31e-32a6-11d3-8f4c-00a024bafceb</METABASE:doc_id>
</METABASE:Document>
<METABASE:Document>
  <METABASE:doc_id>786d7024-36f7-11d3-bec9-00a024bafceb</METABASE:doc_id>
</METABASE:Document>
```

representing the UUIDs of all documents added since July 4, 1999. The subscriber would then make the following request:

```
GET /MB.cgi?table=Document&xml=1&uuid=c619b31e-32a6-11d3-8f4c-00a024bafceb&id_type=doc_id
```

Which would generate the following response:

```
<?xml version="1.0" standalone="yes"?>
<METABASE:Document>
  <METABASE:doc_id>c619b31e-32a6-11d3-8f4c-00a024bafceb</METABASE:doc_id>
  <METABASE:doc_desc>"Waldstein" Sonata</METABASE:doc_desc>
  <METABASE:creation_date>Mon, 05 Jul 1999 07:07:22 GMT</METABASE:creation_date>
</METABASE:Document>
```

The subscriber would make the same request for each and every UUID that was returned by the publisher.

While it is true that this system is highly simplistic and tends to return results with rather coarse granularity, it must be remembered that it is not the purpose for the CGI interface to act as a search engine. Rather, it is its purpose to be used for retrieving the metadata information that been added or updated since the last replication. More complex searches are possible with customized client tools.

**Metadata Examples**

Let's create a sample collection of metadata that could hypothetically be represented. Let's assume that we are creating metadata for a small repository of music. First, we start with categories, let's say we create the categories classical, and under classical, the category concertos. We will need to create two UUIDs for the categories: a7b6d5c0-
374f-11d3-864f-00a024bafceb and ba2e9bb6-374f-11d3-b14a-00a024bafceb. This in turn translate into the following SQL:

```sql
insert into Categories(cat_id, cat_desc, parent_cat_id, creation_date)
values( 0xa7b6d5c0374f11d3864f00a024bafceb, 'CLASSICAL', NULL, 05 Jul 1999 07:07:22)
go
insert into Categories(cat_id, cat_desc, parent_cat_id, creation_date)
values( 0xba2e9bb6374f11d3b14a00a024bafceb, 'CONCERTOS', 0xa7b6d5c0374f11d3864f00a024bafceb, 05 Jul 1999 07:07:22)
go
```

We then want to add two concertos to the metadata repository. We therefore create two other UUIDs, 5edc1620-3750-11d3-91d0-00a024bafceb and fedb0744-3750-11d3-8e53-00a024bafceb.

```sql
insert into Document(doc_id, doc_title, doc_desc, creation_date)
values(0x5edc1620375011d391d000a024bafceb, 'Concerto for Piano and Orchestra no. 20 in D minor', 'K. 466', '05 Jul 1999 07:07:22')
go
insert into Document(doc_id, doc_title, doc_desc, creation_date)
values(0xfedb0744375011d38e5300a024bafceb, 'Concerto for Violin and Orchestra no. 3 in G major', 'K. 216', '05 Jul 1999 07:07:22')
go
```

We would also like to add a new relationship to the data model—that of a composer.

```sql
CREATE TABLE Composer (
    composer_id          binary(16),
    composer_name     char(255),
    creation_date        datetime,
    PRIMARY KEY (composer_id)
)
go
CREATE TABLE Document_Composer (
    composer_id          binary(16),
    doc_id               binary(16),
    creation_date        datetime,
    PRIMARY KEY (composer_id, doc_id),
    FOREIGN KEY (composer_id) REFERENCES Composer,
    FOREIGN KEY (doc_id) REFERENCES Document
)
go
```
So we will add Wolfgang Amadeus Mozart as a composer, and add the relationships to the two documents we have created. Of course, we will need to create another UUID: 99159128-37c2-11d3-b2e7-00a024bafceb.

```sql
insert into Composer(composer_id, composer_name, creation_date)
values(0x9915912837c211d3b2e700a024bafceb, 'Wolfgang Amadeus Mozart', '05 Jul 1999 07:07:22')
go
insert into Document_Composer(composer_id, doc_id, creation_date)
values(0x9915912837c211d3b2e700a024bafceb, 0x5edc1620375011d391d000a024bafceb, '05 Jul 1999 07:07:22')
go
insert into Document_Composer(composer_id, doc_id, creation_date)
values(0x9915912837c211d3b2e700a024bafceb, 0xfedb0744375011d38e5300a024bafceb, '05 Jul 1999 07:07:22')
go
```

One should also create subjects to attach to the documents, possibly "Violin Concerto" and "Piano Concerto". The subjects will be assigned unique UUIDs, of course, allowing a subject's description to be called either "Violin Concerto" or "Violinkonzerte"—it would not matter to the metadata system. The UUID is the subject, the textual description is just that—a description. I will not produce SQL statements of inserting subject values; it suffices to say that it would be much like attaching composers to documents.

The most salient point about the Metabase system is the difference between a "Document" and a "Doc_Copy". If we had two different files for Piano Concerto No. 20, one in the PCM format, and one in the MP3 format, we would have two copies of the same document. Of course, we need to create an UUID for the copyright text (14ca54b8-37df-11d3-acb6-00a024bafceb) and another UUID for the reviewer (219497c6-37df-11d3-800e-00a024bafceb). We also need two UUIDs for the two copies of the document: 81f57b76-37df-11d3-b973-00a024bafceb and 86529fc8-37df-11d3-a12a-00a024bafceb. The SQL for adding the rights and reviewer to their respective tables will not be shown for brevity. The SQL statement for the document copies will look like the following:

```sql
insert into Doc_Copy (copy_id, doc_id, right_id, language, reviewer_id, country, format_mime, review_date, url, creation_date)
values( 0x81f57b7637df11d3b97300a024bafceb, /* copy_id */
0x5edc1620375011d391d000a024bafceb, /* doc_id */
0x14ca54b837df11d3acb600a024bafceb, /* right_id */
NULL, /* this is not in any particular language */
0x219497c637df11d3800e00a024bafceb, /* reviewer_id */
NULL, /* country code does not apply */
'audio/x-wav',
'05 Jul 1999 07:07:22', /* review date */
http://www.foo.edu/bar/81f57b7637df11d3b97300a024bafceb.wav',
```
Summary

The Metabase metadata system is a fairly complex one. This makes it rather unsuitable for simple collections such as source code and binaries. In fact, using Metabase for a simple collection of files would be a complete overkill. For example, due to the many to many relationship that exists between documents and categories, any document can exist in multiple categories. The Metabase system would require a considerable amount of maintenance for it to be truly valuable. Metabase is almost completely reliant on having a group of "experts" who act as editors or reviewers and decide whether a change gets committed to the metadata or not.