Open Data Management & Cloud Exam Project

Patrick Indri

September 1, 2008

Introduction

Aim of the project

The aim of this project is to investigate audio file archiving for music. In particular, a data model will be presented and its implementation prototyped. The model will focus on external descriptive metadata, handling basic technical metadata as well. Data discovery/access and interoperability will be discussed and related to the proposed model. Additionally, cloud solutions and practices for long term archiving and data preservation will be considered and briefly examined. On what follows, "metadata" will be used to refer to *external* metadata unless otherwise specified.

Data resource

The project will not be based on an actual dataset. Instead, it will generally address music files. Audio files for music display a great variability, which must be taken into account. In particular, music files can have different:

- file formats;
- encodings (for a given file format);
- metadata containers and contents.

In fact, no standard exists (or, if it does, is commonly used) for neither audio file formats (and their encodings) nor metadata models. This intrinsic variability must be handled by the model and will be further discussed.

Metadata standard for audio files

There is no widely used and standardised metadata model specifically tailored for audio files. Audio (and audio-visual in general) archives have historically been managed with ad-hoc solutions[1]. Audio archives are challenged in the first place by the transition from analog to digital formats and once digital formats are obtained, standardized systems to catalogue, describe and generally handle the resulting files do not exist. Standards in the field are fragmented and achieved by consensus and usually deal with particular areas of interest. However, the following three metadata standards are usually considered when dealing with audio collections.

- Dublin Core is a small set of vocabulary terms that can be used to describe digital resources. Introduced in 1995 and initially regarded as an impractical solution because of its rigorous simplicity (15 metadata terms in the original specification), Dublin Core has since been diffusely adopted, well beyond music audio files. Focusing primarily on descriptive metadata, it can create basic descriptions of digital resources and is intended to be expanded and extended for specific uses. None of its elements are mandatory and all are repeatable. Extensions of Dublin Core can achieve a finer level of detail.
- The EBUCore metadata set is based on and compatible with Dublin Core. Developed by the European Broadcasting Union it specifically handles radio and television broadcasts. In its intentions, it takes into account the latest developments in the Semantic Web and Linked Open Data communities. It expresses both technical and administrative metadata in great detail and it is well suited for a wide range of broadcasting applications (including archiving, exchange and production). The resulting data model is significantly more complex than Dublin Core.
- The METS (Metadata Encoding and Transmission Standard) is a metadata standards which is able to encode descriptive, technical and amministrative metadata for digital objects. It handles the hierarchical structure of digital library objects ad provides a means to combine elements of different schema into a single record. METS itself does not prescribe a vocabulary, in order to achieve greater interoperability. It takes a sensibly greater effort to create and maintain.

It is clear how different metadata standards offer different levels of detail and flexibility. For the purpose of this project, metadata model will be designed from the ground up.

[1]encicl arc science

Data Model: design

What should the data model be able to represent?

A model for audio data resources should be able to represent songs (and their possible different versions) and their groupings (in albums, compilations or other releases). Moreover, it should model authoring for artists (singers and lyricists) and other professional figures such as music producers. The model should handle basic technical metadata as well. Additionally, relations between different classes (e.g., a work is part of another work, two artists cooperated for a release) should be handled as well.

A more detailed description of the model entities/classes

The proposed model consists of the following four main classes:

- The smallest object (in terms of granularity) is a **recording**, which represents a uniquely identifiable version of a song, that is the actual audio data. ISRC is a standard identifier for *sound recordings* and can be associated to each recording. A recording has title, ISRC, duration, artist credit and a link to the physical data resource (the audio file). Additional technical metadata for the data resource can be provided.
- Different recordings of the same song refer to the same **work**, which embeds all the different versions of a song. It represents the intellectual creation and can be put in relation to different recordings, different artists or other works. ISWC is a standard identifier for *musical works/compositions* and can be associated to each work. A work has name, ISWC and links to correlated works/artists/records.
- Commercial issues of works are grouped into a **release**. Each release contains a track-list for the recordings it contains and has a title, an artist, information about the label and a form of identification (e.g., a catalogue number provided by the label).
- The paternity of works belongs to **artists**, which represent single musicians, groups of musicians or other professional figures (e.g., producers, lyricists, sound engineers). ISNI is a standard identifier for public entities of contribution to media content and can be associated to each artist. Artists have name, type and ISNI.

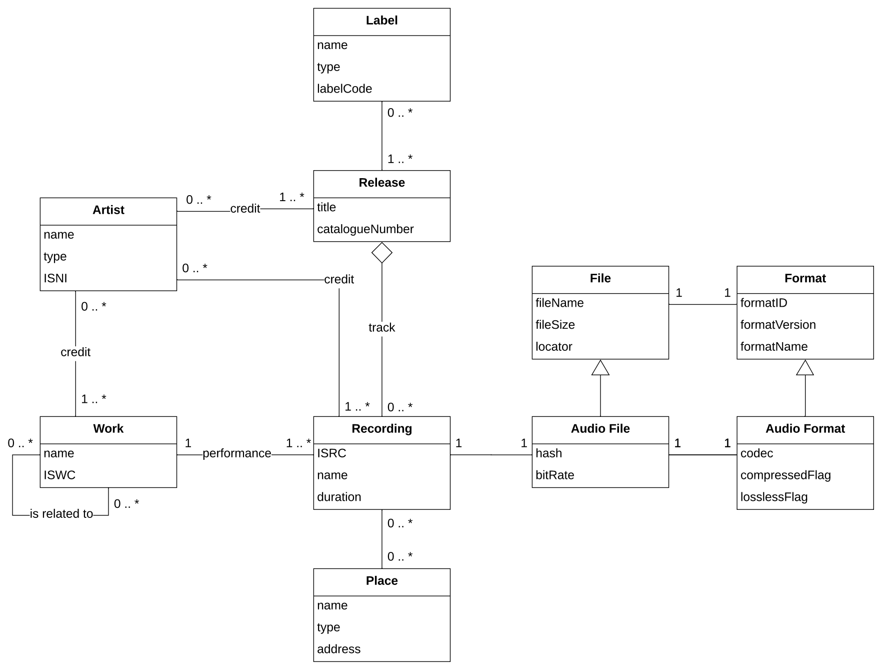
Additionally, the following secondary classes are defined:

- **File**. Base class for all the files. A file has name, size and a locator (that is, a link to the actual resource).
  - **Audio File**. Adds file details for audio files, has a hash and a bitrate.

- **Format**. Base class for format metadata. A format has ID, version and name.
  - **Audio Format**. Adds format metadata details for audio files, has codec, compression flag and lossless flag.
- **Label**. Holds information about imprints and record companies that take part in the production and distribution of a release. Label can uniquely identified by their label code.
- **Place**. Specifies the place where music is performed, recorded, mixed, etc.

UML model

In the following model square endpoints denote aggregation and arrow endpoints denote inheritance.



The proposed design assumes each recording, work and release to be associated with a credited artist (i.e., their participation to the *credit* relationship is total). This assumption can be satisfied defining special purpose artists to deal with unknown authors, traditional songs, etc.

Data Model: implementation

Choice of implementation

Concerning available implementation choices for the proposed model, two macro-alternatives have been taken into account: relational databases (RDB) and XML databases.

Relational databases offer a trivial way to enforce constraints between entities using primary keys and foreign keys. For instance, this would allow to properly model the constraints between releases and artists (i.e., a release cannot be credited to an artist whose name is not present in the “artist” relation). Additionally, indexing is easily supported and RDBs have a widespread use. Moreover, RDBs handle many-to-many and many-to-one relationships with ease (consider the artist-release and the work-recording relationships). However, relational databases manifest a reduced flexibility due to their strict integrity constraints and, once instantiated, are moderately difficult to expand.

On the other hand, XML databases (document-based databases in general) are meant to be a more flexible means to store data, integrity not being their first concern. For a given XML schema, an XML documents can usually contain partial/incomplete information. This is appealing for the task at hand since not all the metadata for a given release may be available. However, many-to-many and many-to-one relationships are harder to handle.

For this particular project, flexibility has been privileged: the model implementation will consists in a XML schema definition (XSD). It should be noted that a PostgreSQL implementation for a similar base model is provided by MusicBrainz, a collaborative music metadata project.

XSD

An implementation of the proposed model can result in XSDs with varying degrees of flexibility. EBUCore XSD provides extreme flexibility and can handle both audio and visual broadcasting resources. Such great flexibility (a necessity, given the wide range of resources EBUCore is able to descrive) is deemed not to provide enough structure for this particular project. At the opposite side of the spectrum a strictly hierarchical XSD, where each document must contain “releases” made of “recordings”, might lead to a overly strict model implementation (in this case, a RDB implementation would provably serve the aim of this project better).

Drawing from the considerations of the previous paragraph, the proposed XSD should:

- Provide, where possible, a refinement of the Dublin Core elements, aiming for interoperability;

- Find a balance between a hierarchic structure and the possibility ti accept "partial" entries (e.g., a document that contains only an artist must be handled by the XSD);
- Handle relationships between different elements with reasonable detail.

The resulting XML schema will now be discussed.

On relationships

The UML model presents several many-to-many and many-to-one relationships which have been implemented with different approaches:

- Recording-Place and Release-Label (many-to-many). This relationship has been simplified by defining a Place/Label to be one of the elements of a Recording/Release. This reduces expressiveness: Places and Labels cannot be stand-alone entities. This solution is acceptable since Places and Labels are secondary entities.
- Recording-Audio File (one-to-one). Each Recording is forced to have exactly one Audio File element (using `minOccurs="1"` and `maxOccurs="1"`). The other constraint is enforced by the existence and uniqueness of a `fileID` attribute.
- Audio File-Audio Format (one-to-many). Each Audio File is forced to have exactly one an Audio Format. Audio Format is not a stand-alone entity.
- Work-Recording (one-to-many). Works have the `hasPerformance` relation element, which links a Work to at least one Recording ISRC. On the other hand, Recordings have the `isPerformanceOf` relation element, which links a Recording to exactly one Work ISWC.
- Work-Work (many-to-many). Works have the `hasRelatedWork` relation element.
- Release-Recording (one-to-many, composition). Releases have the `hasTrack` relation element, which links a Release to at least one Recording ISRC.
- Work-Artist, Recording-Artist and Release-Artist (many-to-many). This relationships have been modelled using the `hasArtist` relation element.

All the constraints (cardinality and participation) have been modelled.

On references (and between different documents)

In EBUCore relations are implemented via REFID and ID. While this choice allows for an easy implementation and provides continuity with DTD, XSD offers a much more flexible KEY/KEYREF syntax: identifiers and references are selected using XPath expressions. Keys and keys references (can) act like the primary and foreign keys of a relational database. Additionally, keys can be multi-field (that is, an item can be identified and referenced using more than one field).

Both ID/IDREF and KEY/KEYREF approaches are flexible enough to describe the relations of the UML model; for presented reasons, the KEY/KEYREF paradigm has been adopted in this schema. It is worth nothing how neither of the approaches allows across-document relations and constraints. XSD integrity constraints are indeed intended for use within a single document. There exist XSD extensions such as W3C's Service Modeling Language or other rule-based validation languages such as Schematron that do handle inter-document constraints: these possibilities have not been addressed.

XML example

The following fragment shows a valid instance of a work. It contains both an `id` to identify the work and an `idref` to reference a related recording. Furthermore, it shows how the `label` and `description` attribute can be used to describe related items.

```
<work>
  <ISWC id="ISWC_T-000.000.000-A"></ISWC>
  <title lang="en">
    <dc:title>Test Work</dc:title>
  </title>
  <hasArtist label="William Wilson" description="Singer"></hasArtist>
  <hasPerformance label="Test Recording" description="Studio Version">
    <relationIdentifier>
      <ISRC idref="ISRC_AAAAA0000000"></ISRC>
    </relationIdentifier>
  </hasPerformance>
</work>
```

A more extensive XML example document has been provided alongside the XSD.

Possible model expansions

The model could be further expanded implementing the following features:

- Sort names, useful for displaying and ordering results. Usual practices include placing surnames first (e.g., Ólafur Arnalds will have "Arnalds, Ólafur" as a sort name) and placing articles last (e.g., The National will have "National, The" as a sort name);
- Images, to store album cover or artist portraits. The File and the Format classes can be extended to deal with image formats. Images could have relationships with Releases, for album covers, and Artists, for artist portraits.

Interfaces and services

A full fledged service that adopts the prototyped XSD is out of the scope of this project. However, some possible services related to the resource discovery and accessibility will now be discussed.

The two main goals an actual implementation of this project should achieve are: music query (i.e., a music search/download service) and database update (i.e., a service devoted to the addition of new documents to the database). Both this services are related to the data storage, which will be briefly discussed as well.

Data discovery

The most fundamental service an archive should provide is a search/filter service.

In order to make data accessible in the first place, a web service should be implemented. Users should have network access the archive, and the archive web page should be identified by a URL. The filtering service should offer the typical search fields for music queries: users should be able to query authors, works, recordings and releases. Luckily, these usual query fields correspond exactly to the main classes of the presented model. Thus, searching for a specific artist would parse the `artist` elements only. Queries would most probably be free-text queries on the various classes.

On the back end, queries on XML files could be carried out using XQuery, a query language built on XPath (and a W3C Recommendation for XML queries). Since the data model is known, and has a reasonably strict structure, simple queries could be trivially implemented. For instance, XPath queries are supported by the `xml.etree.ElementTree` Python module.

Traditional information retrieval techniques could be used to substantially improve the results of a query. First of all, spelling correction techniques could be employed. The *Levenshtein distance* offers a simple approach to spelling correction: if the query returns no record, and thus the query possibly contains a spelling error, the closest results in terms of number of edit operations are retrieved. A query for “Pink Floid” will probably return no results. However, “Pink Floyd” has a unitary edit distance and will be consequently returned as a result. Alternatives approaches would involve using k-gram distance and the Jaccard coefficient for queries. This simple improvement would certainly favour data discovery. Additionally, a popularity value could be used to rank results: the user would be answered with the most popular (in terms of queries or downloads, for instance) items.

Indices should be built for better performance. XML indexing is a younger field of research, if compared with RDB indexing, and suffers from some decisive disadvantages. As with any index structure, edits are expensive since they require to rebuild the index. In particular, construction costs affect XML indexes severely. Moreover index size does not scale well with the size of XML files. The trade-off between increased performance and increased disk space should be deeply investigated if the project were to be put into production.

Data access

Once the data has been located, it should be easily accessible via download. The user should be able to separately download different versions (i.e., different file formats) of the same resource, if they are available. Ideally, a preview of the resource should be available as well: users should be able to listen to a certain song in their browser without downloading the file itself. It must be pointed out that this is possible only if the file storage allows sequential access to files. As an additional service, a user might want to download all the songs credited to a certain artist: a mechanism exploiting XPath queries to build on-demand compilations should make this service possible.

Data annotation

In all previous sections of the project a strict division between data and metadata has been enforced: the XSD models the metadata structure which holds references to audio files that are separately handled. However, audio files themselves can be embed metadata. Unfortunately, no standard metadata container for audio files knows widespread use.

Whether metadata *should* be embedded into audio files is a separate question. A minimal amount of metadata (the *catastrophic* metadata) is necessary to uniquely identify the resource in the event of a disastrous disassociation with its corresponding metadata file. Storing a unique identifier o a brief description would be sufficient. Association with an external metadata file is generally necessary, since embedded metadata have a limited coverage and are difficult to maintain and index.

What follows focuses on commonly used audio file format; they will addressed again in the interoperability and preservation section.

- ID3 (and particularly the extended ID3v2) is the *de facto* standard for Mp3 files, the most common and widely used audio file format. ID3 handles all the common music metadata tags in a structured fashion and can store all the metadata of the proposed model. Additionally, it supports image tags to store album covers and similar content. The `mutagen` Python library can be used to handle audio metadata. It is worth citing MusicBrainz Picard, a free and open-source software application that can automatically identify and tag audio files using the MusicBrainz database. Despite its extensive use, ID3 was designed specifically for Mp3 files few other formats support it (WAW being one, but not without compatibility concerns).
- XMP (Extensible Metadata Platform) is an ISO standard commonly used in JPEG images. The standard defines a way to store XMP metadata in a wide variety of file formats, including some audio file formats such as Mp3 and WAW.
- BWF `<bext>` chunks are a recognized archival standard for BWF WAW files and can be declared using XML. However, they offer a very limited

number of fields that are better suited for *administrative* metadata rather than descriptive ones.

- Vorbis comments are metadata containers used in the Vorbis and FLAC audio file formats. They consist in unstructured key/value pairs in the format `fieldname=data`, and the same field can be repeated multiple times. This approach is in contrast with the highly structured ID3 approach.

Since no metadata container is supported by all the most common formats, the choice for a specific container heavily depends on the file format. Whatever the choice, software and programming languages that can handle metadata manipulation are available (e.g., the previously cited `mutagen` library) and could be used to embed/map the content of the proposed model in the audio files.

Storage and cloud solutions

Following the proposed model, an actual music archive would be required to handle both XML documents (the metadata) and audio files (the data): they will be now discussed separately. Cloud (storage) solutions will be addressed as well.

Storing XML files

XML documents can fall in two (blurry-edged) categories depending on the rigidity of their structure: data-centric, more structured, and document-centric, more irregularly structure. The proposed XSD would probably result in data-centric documents, since some rigidity and integrity constraints are, in fact, enforced. Conceptually, two XML storage approaches exist: the first one requires to build a map for the XML model, mapping it into a database model, the second one maps XML documents into pre-existent database structures that can store any XML. The first approach is typical of XML-enabled databases, the second of XML-native databases.

Usually, data-centric documents can be more efficiently and conveniently handled using XML-enabled databases. This conclusion is in direct contrast with the XPath query approach of the data discovery section (XML-enabled databases can be queried using SQL). On the other hand, a XML-native database would keep the XQuery approach but would probably scale worse.

In fact none of the three most used XML-native databases (eXist, Sedna and BaseX) offer the scalability properties XML-enabled NoSQL databases have. In particular, they are limited to share-nothing distribution and not all of them offer redundancy services; this shows how they are still somewhat unripe alternatives. Non of them is, in conclusion, horizontally scalable. BaseX, for instance, is light-weight by design and many share-nothing and separate BaseX instances are necessary when the size of the database increases. These alternatives offer no cloud solutions at all.

On a side note, the proposed XSD would note scale well itself. At its current state, the schema would practically force all metadata to be stored in a single, large XML file. This is a necessary price to pay if some consistency is enforces (i.e., if KYE/KEYREF is used). As previously stated, W3C's Service Modeling Language and Schematron can handle across-document constraints, softening this design issue.

Storing audio files

As the number of stored songs increases, the storage space for the audio files will increase linearly. Different storage solutions can be adopted as required storage space increases.

First of all, audio files should not be stored as part of the metadata database. Notwithstanding the support for BLOB (Binary Large Object) data many SQL databases offer, storing a pointer to the resource is a more flexible and scalable alternative. Including BLOBs would result in larger databases, high memory usage and slow query response. This very approach is adopted in the proposed XSD.

For small archives (up to *few* terabytes), simply storing audio files in a remote file system would probably be a cost and time effective solution. Concurrent access should be managed, with the most trivial solution being single user access. As the archive size grows, a distributed NoSQL database would become necessary. Both MongoDB and Cassandra offer Database-as-a-Service solutions.

A successful music archive could need to manage many concurrent accesses (both of data and metadata). Spotify, a very large music streaming service, offers some insights on how to mange a large number of files with many accesses. Up to 2017, Spotify used Cassandra for most of their services. Cassandra was chosen over PostgreSQL because it guarantees better replication and resilience to failures, while allowing easy communication across data sites. These characteristics would perfectly fit the requirement of a large audio file storage. More recently, Spotify completed the transition of its backhand to a Google Cloud Platform.

Cloud solutions would actually be very convenient even for small archives. Amazon S3, for instance, offers cheap storage with versioning capabilities (which would be useful for file updates) and seamless orphan files handling. Redundancy and backups are triggered via command line and the storage size scales effortlessly as the archive gets larger. Concurrent access and access permission can be handled as well.

To summarize, audio file storage would deeply benefit from cloud solutions, particularly as the archive size increases.

Interoperability and preservation

- Interoperability (look at slides), open file formats, bwf, unique identifiers;
- S3 Glacier and Glacier Deep Archive.
- **OAIS**: sketch model.

Software and tools used

New content

References

Project notes

TODO:

- address type;
- check UML cardinality;
- data models for discovery;
- more details on the preservation section;
- standard for archiving: WAW 96khz, 24bit or FLAC, with MP3 download;
- check definition of dc refinement