Преглед НЦД 13 (2008), 27–36

Saša Malkov
(Faculty of Mathematics
University of Belgrade)

ON PHOTOGRAPH LIBRARY DESIGN – GROMAN LIBRARY

Abstract: The development of digital photograph libraries introduces many technical problems. To develop a useful solution, it is necessary to make it efficient and simple. On the other side, to fulfill its primary purpose of information preserving, the solution has to be based on general principles and to support the gathering of as general data and metadata as possible. A general model of digital photograph library is presented in this paper. The common problems are discussed and appropriate solutions are proposed. The discussion and propositions are based on experiences of development of Groman photograph library.

Key words: digital library, photography, design

1. Introduction

The development of a contemporary online photograph library faces many different problems. Among the most usual problems are: wide variety of digital photographs formats; wide image size range; a variety of sources; the necessity for many different instances of single photograph; multiple classifications of photographs and many different client interfaces for all kinds of users, from archival officials to public online users.

A discussion on designing a digital photograph library is presented here. The initial design was introduced for development of Groman Library. Groman Library consists of XIX century photographs made by Russian military photographer I. V. Groman. The photographs present Belgrade, Jagodina, Paraćin and other Serbian towns, as well as battlefields. These photographs are among the first pictorial evidences about Belgrade and Serbia in the XIX century [1].

During the development of the Groman Library, it became obvious that most of the problems present with this specific library are common with many other photograph libraries and archives. Thus, some generalizations of the design elements are introduced, in order to provide a more general photograph library design template. However, while the generalization was an important designing principle, at the same time the model is kept as simple and flexible as possible. Different kinds of additional functionalities are not included, in order to avoid any complexities, which are important for specific libraries only.

In the following sections, the problems and proposed solutions are presented. Finally, a relational database scheme for the model is presented.

2. Problems and Solutions

2.1. Variety of Digital Formats. A photograph may be stored in many different digital formats. The characteristics of the formats make each of them appropriate for some of the possible purposes, but no single format is perfect. If a digital format preserves the highest possible original image quality, without any data loss, then it provides no significant compression and requires more memory and higher communication bandwidth. On the other

side, if a format features a high compression, then it allows for smaller memory and bandwidth requirements, but introduces some image quality loss.

If a digital library is intended for many different purposes, it is necessary to support many different formats for each of the photographs in the library. However, sometimes it is not possible or practical to provide each of supported digital formats. For example, if a photograph is originally made using a digital camera and saved in a format with data loss compression, it would be a waste of space and time to add to the library the same photograph in some additional lossless format, because the quality would be the same. On the other side, if the photograph is additionally processed, then it is reasonable to save it in both kinds of formats, with and without quality loss.

2.2. Image Size Range. It is not easy to select a preferable image resolution for a library. A higher resolution allows (and often is necessary) for better quality, but results in larger files and more difficult image content manipulation. On the other side, lowering the resolution results in smaller images and easier manipulation, but may limit the usability.

It is desirable to support many different resolutions for each of the photographs in the library. At least three different sizes are recommended:

- Original size – the highest possible resolution; if image is scanned, then the highest possible resolution is to be used. This resolution is highly influential on future use of the library, and should not be compromised. However, it is of no use to select a scan resolution that is much higher then the quality of original analog sources.
- Screen size – a resolution optimized for presentations on computer display, for preview purposes, including Web, presentations and other applications. The final image size should be near the estimated display resolutions. For example, in the Groman library we decided to use image width up to 1200 and height up to 800 pixels. The choice of this resolution is not as important as the previous one, because it is always possible to automatically resize all original images to a new screen size.
- Thumbnail size – a resolution optimized for image lists, when many different images are to be presented in the same time, for browsing or searching purposes. The image size depends on estimated applications requirements. In general, it should be at least three times lower than screen size. For example, in the Groman Library we used images with width up to 300 and height up to 200 pixels. Like with screen size, this resolution choice is easy to customize at any time. Moreover, if the library is used for many different applications, it is reasonable to support many different thumbnail sizes.

2.3. Multiple Instances. A feature that is often necessary, and almost always useful, is a support for multiple different instances of the same photograph. The motifs include the presented variety of digital formats and different image resolutions, as well as many other significant factors, like different scanning techniques, details extraction and many others. The proposed solution is to introduce two different model entities: a photograph and a photograph instance. UML model [2] of the solution is presented in Figure 1.

A photograph (class *Photo*) is a library item, while a photograph instance (*PhotoInstance*) is a single digital image content preserved in the library. Each photograph instance is related to a single photograph and has a specific digital format (*DigitalFormat*) and other important attributes. The instance attributes include *InstanceType*, which is used for instance classifications. Possible instance types are "*original*", "*thumbnail*", "*detail*" and others.

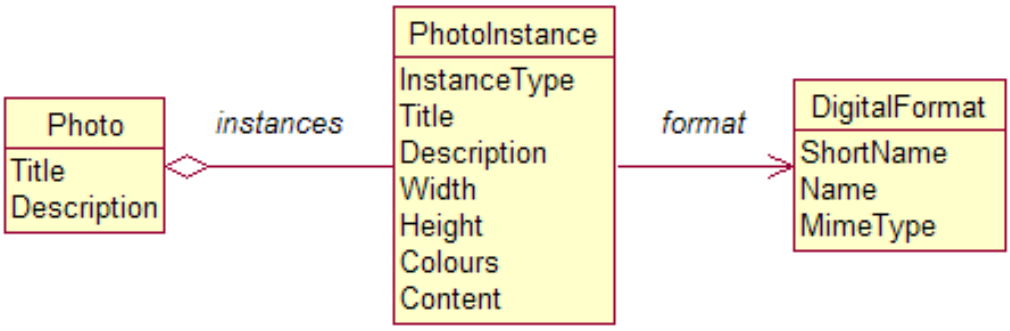

Figure 1. UML class diagram: Photograph instances

2.4. Multiple Photograph Classification. One of the main library features is good content classification. A basic classification assumes that each item has to be assigned to a category from a list of categories. However, contemporary libraries are expected to provide more than that. Two significant improvements are suggested: dynamic category hierarchy and support for multiple photograph classification.

A category hierarchy assumes that all categories are organized in a tree-like hierarchy:

- one of the categories is marked as "root category";
- each other category is related to single "parent category".

Such organizations allows for efficient browsing of the library content. However, some additional care is to be taken:

- If a category is "parent" for too many "child" categories, the browsing may become inconvenient. The number of "child" categories should be uniform across the hierarchy tree as much as possible.
- If there are too many items assigned to some categories, or if there are many categories with very few items assigned, then it is not possible to browse efficiently. The total count of categories (and the size of the category hierarchy) should be taken in relation to the library content.
- It is usually not possible to satisfy both of the previous two points. Depending on the specific conditions, these two rules have to be harmonized as much as possible.

A dynamic category hierarchy assumes the possibility of dynamic editing of the hierarchy tree during the library lifetime. In practice, the library is not loaded uniformly in different sections and that results in categories of non-uniform sizes. The only practical solution is to provide tools for dynamic editing of the category hierarchy.

As a library grows, the library category hierarchy grows also. That leads to narrowing of single categories semantics. As a result, it may be hard to precisely classify an item to a single category. The solution is to support the classification of a single library item in many different categories. Thus, if it is not possible to classify an item to only one category, then such item may be classified to each of the categories to which it is related.

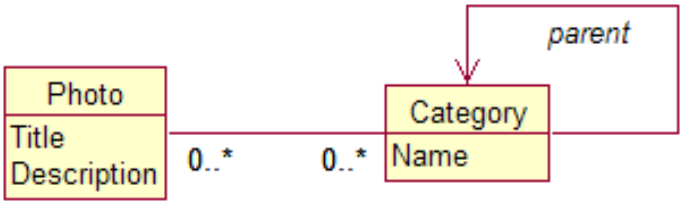

Figure 2. UML Class Diagram – Photograph categories

A UML model for photograph classification is presented in Figure 2. The presented model supports the construction of category tree, and multiple classifications of single photographs. Each category (*Category*) has a name and may have a parent category.

2.5. Metadata. Each photograph in the library is described by some metadata. Some of the attributes are common, but some may be specific. Common attributes include image size, image resolution, time and location, photographing technique, and other. Classes Photo and PhotoInstance are kept with small number of attributes because of the clarity of the presentation. However, many more common attributes related to photographing deserve to be added to these classes, like aperture value, focal length, shutter speed and others.

The library content comes from different sources, which provide different kinds of metadata. Even in a small photograph collection it is often possible to find completely different metadata collections. To preserve the completeness of original resources, it is very important to support any available kind of metadata. The high quality metadata is necessary to exploit the library content. It is highly recommended that metadata may contain any information related to library items. In other words, no single metadata should be discarded because it is not supported by the library model. Otherwise, a lot of significant data would be disposed.

One solution is to preserve these non-standardized metadata in a form of unstructured (i.e. non-checked) XML documents. Another is to provide an appropriate native library design. We would suggest the usage of appropriate library design, because it usually offers better searching features.

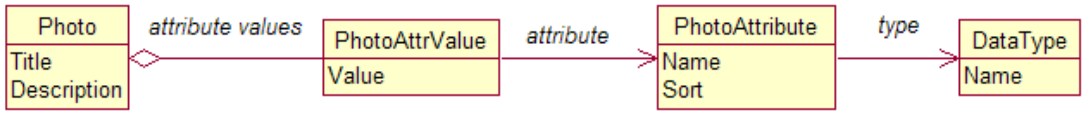

Figure 3. UML Class Diagram – Photograph metadata

A simple metadata model is presented in Figure 3. Each photograph (*Photo*) may have many attribute values (*PhotoAttrValue*). Each attribute value is related to an attribute (*PhotoAttribute*), and each attribute has a data type (*DataType*). The model is relatively simple, and understandable, but it is not complete and flexible enough.

This model does not explain how different types of metadata are represented. An attribute of string type is not the same as an attribute of integer type. One solution is to modify *PhotoAttrValue* to include values of all supported primitive types. However, the primitive types are not universal enough, and there always can appear an attribute of non-primitive data type. The better solution is to rely on type conversions and to represent all data types using their string representation. Both the application and the database may carry out the conversions.

Another problem is related to structured or sequential attributes data, which are not supported by this model. It is common to have some sequential metadata. For example, if a sequence of image processing operations is to be described as metadata, it is better to natively support such data then to introduce enumerated attribute names, like “processing1”, attribute “processing2” and so on. One solution is to provide an additional class for sequential attributes. Another is to add an optional index attribute to already introduced *PhotoAttrValue* class. In this case, we prefer the second solution.

Furthermore, some metadata are related not to photographs, but to photograph instances. For example, if an instance represents only a selected part of a whole photograph, then it is necessary to describe the part position. Or, if an instance represents a processed original image, it is very useful to have the processing description. The solution is to

introduce a class *PhotoInstanceAttrValue*, which is very similar to *PhotoAttrValue*, but is related to *PhotoInstance*, instead of *Photo*. Such modification introduces the full metadata support for both photographs and photograph instances in a highly unified way.

The Figure 4 presents the improved metadata model.

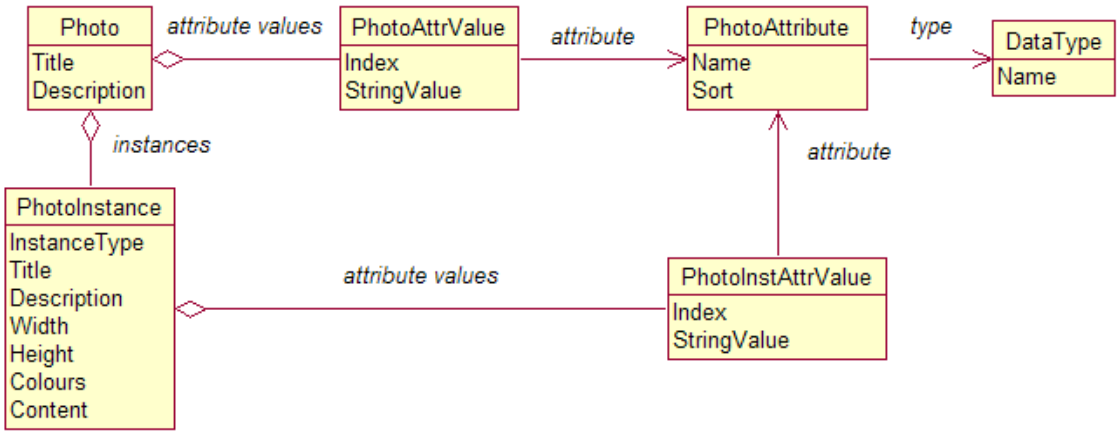

Figure 4. UML Class Diagram – Improved photograph metadata model

2.6. Authors. The Groman Library is specific because all photographs are authored by a single person – I. V. Groman. However, the most of the libraries contain images of different authors. Even more, it is usually not enough to identify the photographers only, but some other kinds of authors and coauthors also: the scene author, costume author, the model and others. Even if we do not need the author data in the Groman Library, here we present an appropriate model, to make the presented model complete.

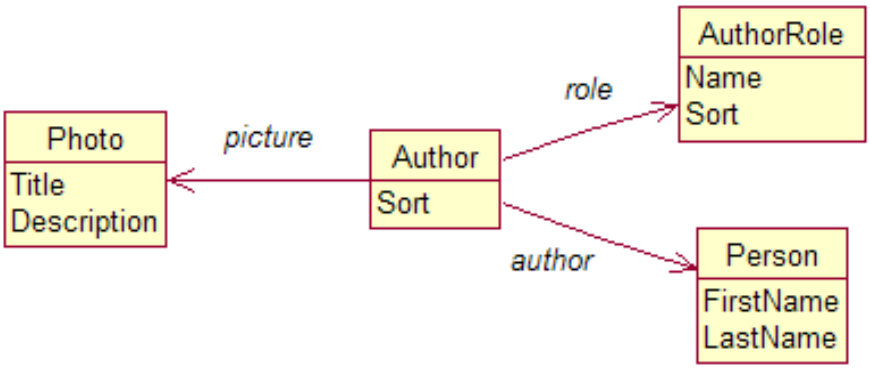

Figure 5. UML Class Diagram – Authors model

The Figure 5 presents the UML model for authoring data. It is quite flexible. An author (class *Author*) represents a single participant in the photograph creation. It is related to a photo (class *Photo*), to an author role (class *AuthorRole*) and a person (class *Person*). Many persons may have the same role for the same photograph, in a given sorted order. The same person may have many roles for a same photograph.

Because of the model clarity, a person is represented by the first name and last name only. Other personal data may be added to *Person*.

If significant image processing is handled on photograph instances, it may be reasonable to consider the instance authoring data, too.

2.7. Different Client Interfaces. Contemporary libraries very often need many faces, some for the experts and some for the public. Different purposes require different usage privileges

and client interfaces. It is usual that completely different applications are used for internal and public purposes. Library databases are rarely specifically customized to specific applications. Such customizations are usually limited to the support for different digital content types and resolutions, which is already discussed in subsections 2.1 and 2.2.

3. Model

Each of the model elements is presented in the preceding sections. The Figure 6 contains the complete UML class diagram for the library.

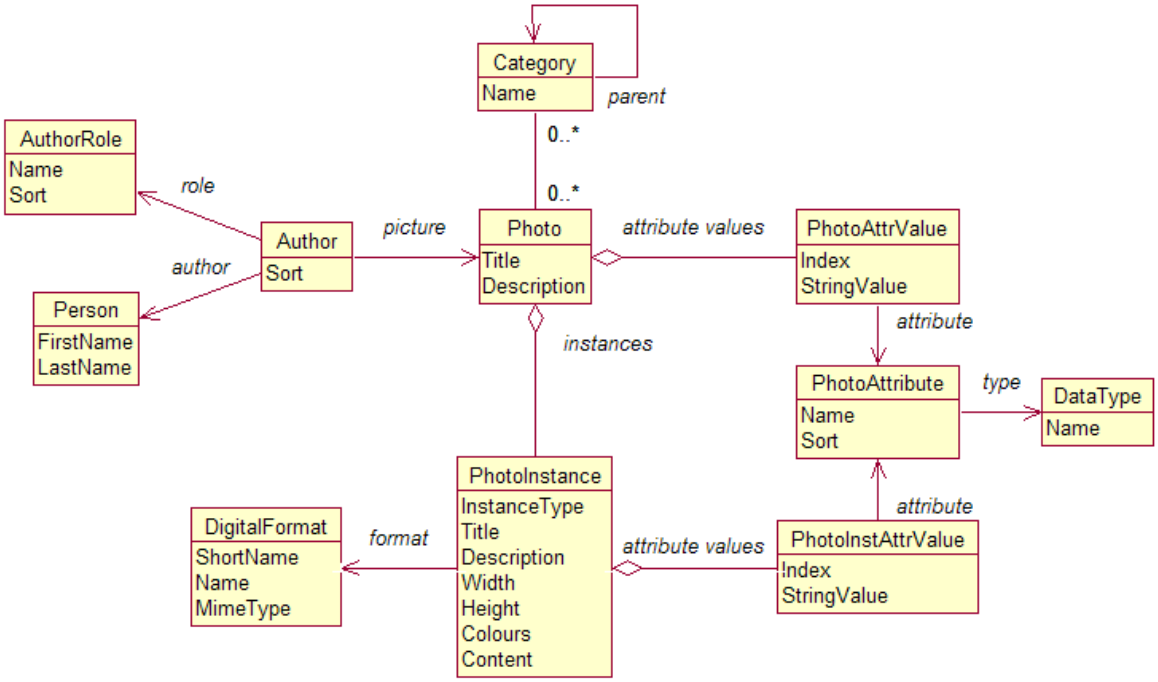

Figure 6. UML Class Diagram – Photograph library

4. Database Schema

The class diagram presented in Figure 6 translates to relational database schema presented in Figure 7. Each of the classes is represented by a table. An additional table *PhotoCategory* is introduced to represent multiple bidirectional association between photographs (*Photo*) and categories (*Category*). The schema is designed for relational database management system DB2 [3]. It is highly compatible to SQL standards [4].

5. Conclusions

The presented model is implemented in the Groman Library. It was proved in practice that it is fully applicable and that provides for both the simplicity and the flexibility.

Each library consists of a library database and one or many applications, which provide different user interfaces to the library. The selection of appropriate software tools depends on the library kind and projected user types. The Groman Library was developed as a prototype for a publicly available photograph library. Thus, the library interface is developed as a Web application. We used functional programming language Wafl [5], which is designed specifically for Web development. For data storage and manipulation we used IBM DB2.

DB2 is one of widely used relational database systems, with advanced binary data and XML features.

```
create table PhotoLib.Photo (
     Id            int not null,
     Title         varchar(200) not null,
     Description   long varchar,

     primary key(Id)
     );

create table PhotoLib.DigitalFormat (
     Id            int not null,
     Name     varchar(100) not null,
     ShortName     varchar(20) not null,
     MimeType varchar(100) not null,

     primary key(Id)
     );

create table PhotoLib.PhotoInstance (
     PhotoId       int not null,
     InstanceId    int not null,
     Title         varchar(200) not null,
     Description   long varchar,
     Width         int not null,
     Height        int not null,
     Colours       int not null,
     FormatId int not null,
     Content       blob(500m) not null,

     primary key( PhotoId, InstanceId ),

     foreign key fk_Photo( PhotoId )
          references PhotoLib.Photo
          on delete cascade,
     foreign key fk_DigitalFormat( FormatId )
          references PhotoLib.DigitalFormat
          on delete restrict
     );

create table PhotoLib.Category (
     Id            int not null,
     Name     varchar(200) not null,
     ParentId int,

     primary key(Id),

     foreign key fk_Parent( ParentId )
          references PhotoLib.Category
          on delete restrict
     );

create table PhotoLib.PhotoCategory (
     PhotoId       int not null,
     CategoryId    int not null,

     primary key( PhotoId, CategoryId ),

     foreign key fk_Photo( PhotoId )
          references PhotoLib.Photo
          on delete cascade,
     foreign key fk_Category( CategoryId )
          references PhotoLib.Category
          on delete restrict
     );
```

```
create table PhotoLib.DataType (
     Id       int not null,
     Name varchar(200) not null,

     primary key(Id)
     );

create table PhotoLib.PhotoAttribute (
     Id       int not null,
     TypeId   int not null,
     Sort int not null with default,

     primary key( Id ),

     foreign key fk_DataType( TypeId )
          references PhotoLib.DataType
          on delete restrict
     );

create table PhotoLib.PhotoAttrValue (
     PhotoId      int not null,
     AttrId       int not null,
     Index        int not null,
     Value        varchar(200) not null,

     primary key( PhotoId, AttrId, Index ),

     foreign key fk_Photo( PhotoId )
          references PhotoLib.Photo
          on delete cascade,
     foreign key fk_Attribute( AttrId )
          references PhotoLib.PhotoAttribute
          on delete restrict
     );

create table PhotoLib.PhotoInstAttrValue (
     PhotoId      int not null,
     InstanceId   int not null,
     AttrId       int not null,
     Index        int not null,
     Value        varchar(200) not null,

     primary key( PhotoId, InstanceId, AttrId, Index ),

     foreign key fk_PhotoInstance( PhotoId, InstanceId )
          references PhotoLib.PhotoInstance
          on delete cascade,
     foreign key fk_Attribute( AttrId )
          references PhotoLib.PhotoAttribute
          on delete restrict
     );

create table PhotoLib.AuthorRole (
     Id       int not null,
     Name varchar(200) not null,
     Sort int not null with default,

     primary key(Id)
     );
```

```
create table PhotoLib.Person (
     Id           int not null,
     FirstName    varchar(100) not null,
     LastName varchar(100) not null,

     primary key(Id)
     );

create table PhotoLib.Author (
     PhotoId      int not null,
     PersonId int not null,
     RoleId       int not null,
     Sort int not null with default,

     primary key( PhotoId, PersonId, RoleId ),

     foreign key fk_Photo( PhotoId )
          references PhotoLib.Photo
          on delete cascade,

     foreign key fk_Person( PersonId )
          references PhotoLib.Person
          on delete restrict,
     foreign key fk_Role( RoleId )
          references PhotoLib.AuthorRole
          on delete restrict
     );
```

Figure 7. DDL – Photograph library relational database

The most of the characteristics of both the problem and the proposed solution are shared with other library types. Almost everything that is discussed here, is applicable not only to photographs but to other kinds of library items, also. The proposed solution is ready to be used solution for images, texts, audio recordings, video recordings, museum artifacts and many others.

References

[1] Ž. Mijajlović, Z. Ognjanović, *A Survey of Certain Digitization Projects in Serbia*, NCD Review 4 (2004), 52–61.
[2] J. Rumbaugh, I. Jacobson, G. Booch, *The Unified Modeling Language Reference Manual*, 2nd ed., Addison-Wesley, 2004.
[3] IBM DB2 Web Site, IBM Corporation, http://www.ibm.com/db2
[4] ISO/IEC 9075: 2008 Information Technology – Database Languages – SQL, ISO/IEC, 2008.
[5] S. Malkov, *WAFL – Functional Programming Language for Development of Web Applications*, Master Thesis, Faculty of Mathematics, University of Belgrade, 2002.

Саша Малков
(Математички факултет, Београд)

О ПРОЈЕКТОВАЊУ ФОТОТЕКЕ – ГРОМАНОВА ЗБИРКА

При прављењу дигиталних фототека аутори се суочавају с више техничких проблема. Да би решење било корисно, потребно је да буде ефикасно и једноставно. Са друге стране, да би фототека могла да испуни свој основни циљ прикупљања информација, она мора да почива на довољно општим принципима и да подржава чување најопштијих могућих података и метаподатака. У раду се представља основни модел дигиталне фототеке. Дискутују се неки од проблема са којима се аутори суочавају и предлажу се прихватљива решења. При разматрању проблема се ослања на искуства стечена током израде Громанове дигиталне колекције фотографија.

mailto:smalkov@matf.bg.ac.yu