TOMS
The Object Management System

by Peter Peerdeman & Timen Olthof
http://www.niftysystems.nl

Abstract

TOMS stands for The Object Management System. The goal of this system is the creation of an open-ended, highly versatile object management system database. The system is designed to enable its users to easily submit and maintain items in their own portfolio-like system and represent their works in various ways. By separating the database and content management from the representation, users are able to use the system to get the object information into their own representations without having to hardcode their content into their application.

Part I
Envisioning the system

1 Introduction

In this project we will create a object management system for people who want to visualize their work. By creating one uniform system, users should able to not only store their works and information on these works, but also query this system in various ways to create dynamic and interactive representations of their portfolio.

To make sure that we create a system that is usable and functional for real end users, we will perform some exploratory case studies on several artists to find out what the functional requirements are for our system. Next to the requirements we will gain more knowledge about portfolios and the contained artistic works in general. This will help us to design the database to be as usable as possible. We thank Siebe Bluijs[1], Peter Lub[2] and Puck Holshuijsen for their cooperation in the project.

After the exploratory case study we will describe the development process of the object management system including the design decisions, software architecture views, rationale and features that could be improved more.

2 Case studies

In this section we examine a few real world artist portfolios, in order to gather more information about what kinds of items the system should support.

2.1 Artist A - Siebe Bluijs

The first portfolio we examine is that of graphic artist Siebe Bluijs. His work ranges from graphic design (books, posters, logo's and more) to illustrations, essays, fonts, paintings, photography and moving pictures. Most of his work is organized in projects, that sometimes contain earlier versions or alternate versions of the same work. Relations between items can also be found between items from different projects. For example, a custom designed font may be used in several projects, resulting in a relation between the font and all those projects.

When enumerating the properties of his works from a portfolio organisation point of view, there are several properties that seem to be distinctive. Not all works do have a real name, but often there is an identifier that indicates the work. Then there is the technique that was used for the particular work, for example oil painting, computer or metal. This is different from the material of the work, which may be for example print, digital or canvas. Of course, in case of a metal object for example, a photo of the object will be stored in the portfolio. This means that the file type will be yet another type associated with (representation of) a work. Finally there is also the medium of the work, which is again a different property. For example, an item with medium 'book cover' may be created using the computer (technique) and have 'print' as its material.

Next, there is the difference between works that have been published and works that do not have been published. Both works that have been published and works that do not have been published may have a print run ranging from 1 to infinity. Of course not all works have a print run. For example video and fonts do not. Finally, Siebe's works can be classified based upon the colors they contain, as well as based on the inspiration source of the works. For example, a painting of John Coltrane is most likely inspired by music.

[1] Siebe Bluijs, http://www.siebebluijs.nl

[2] Peter Lub, http://www.kudsite.nl

2.2 Artist B - Peter Peerdeman

The second portfolio case is one that describes Peter Peerdeman. As a computer science student and musician his works vary from written papers, programs in flash, websites and repositories to mp3 songs and music video's / film video's uploaded to youtube. The current way of displaying his works is an enumerated list of all projects he has done in his student career on a student webpage hosted by the Vrije Universiteit. The works related to music aren't exposed anywhere else, though Peter has had an open directory for music recordings for a while, but closed these due to Google's indexing and huge server loads.

The relations between works isn't made explicit anywhere, though there are some works that are related in a way. For instance, a certain mp3 song may be featured in a music video. Because of the lack of centralized storage there is little information available on the existing works, except for a short description that is displayed with each project in the current portfolio. This however is no problem for a future system, as the rest of the details could be filled into a new system on demand by this user himself.

For a song or a paper, it is sufficient to have only one filename related to each work. However, when storing a programming project's sourcecode, or storing an album of music, multiple filenames are needed that relate to one single work. The system should be capable of making distinctions between one "end result file" for a project and "related files" which can be any number of files.

2.3 Artist C - Peter Lub

Peter Lub is a new generation internet multimedia artist, specialized in flash animations and photoshopped comics. On his website, `http://www.kudsite.nl` he offers his audience a wide variety of short animations (20 seconds), long animations (3 minutes), comics in png format and a blog. He does not really have a separate portfolio, the aforementioned website is basically his showcase which he refers to when people ask about his work.

Though Peter doesn't need a seperate portfolio per se he is interested in the possibility of showing certain content in different ways. For instance, when he has certain images in a portfoliosystem, he would like to show these images in an animated flash banner, but also in a slideshow or gallery on a mobilephone or slow internet connected viewer.

Because of bad experiences with web hosting companies Peter doesn't host his content himself, he hosts all of his pictures at `http://tinypic.com` and all of his video's on `http://www.youtube.com`. The only content that is hosted is his website, which is a Wordpress[3] enabled php website with embedded flash elements. The main issues for using an object management system would be that all files would be hosted on his account, including possible downtime, possible loss of information and the frequent backing up.

2.4 Artist D - Puck Holshuijsen

Puck Holshuijsen is a Multimedia student with a background in rockmusic and filmschool. His current work consists mainly of music of his bands and other projects, full length songs in MP3 format. Also, some of the songs he has created are accompanied by selfproduced videoclips. The videoclips are stored on youtube. The last component of Holshuijsens oevre is a smaller section of photo's, pictures and drawings which are related to both the music and the video's.

Puck's main goal of a portfolio system would be to sort and categorize his work to give a better overview of what he has done. He would like to seperate different categories in his work such as different music albums or different time periods. The date tag given to each item would play a significant role in the periodizing, which enables him to show his work sorted in time.

Holshuijsen has a special interest in relations between works. This could be just a text with a short description of the comparison. Puck would find great use in creating a section of his site in which he compares different works next to each other accompanied by a short description on how the work is relatod or has evolved over time.

[3] WordPress is an open source blog publishing application.`http://www.wordpress.org`

Puck is also familiar with earlier work of Peerdeman and Olthof, and would like to see a seamless integration of PowerRail[4] to present his work in public while giving an actual presentation. In this way, he can manage his work both from a website content point of view as well as presentational content view at the same time.

2.5 Artist E - Timen Olthof

The last portfolio we examine is that of Timen Olthof, computer science and philosophy student, web and interaction designer. His portfolio includes repositories for various courses, written papers, a few short movies and some music compositions. He also wants to include screenshots and links to websites he has built in his portfolio.

Other things that might be included in his portfolios are links (to interesting websites or youtube videos), quotes (from interesting people) or ideas (short texts), as well as book, music and video ratings. It would be interesting to see how tagging these items could create links between these totaly different concepts.

Timen would like to offer multiple presentation forms of the same work on his website, so he would like the system as much presentation-independent as possible.

2.6 Case study conclusions

By questioning the artists and processing these interviews into the textual summaries that were seen in the previous sections, we found some interesting observations, which will guide us in our challenge to build the system according to the requirements of the end users.

Things that we learned about the artistc works, or "items" were that an item may have multiple authors. This makes the creation of items in the system a little more complex but shows the possibility to create interesting possibilities for the end user. We also learned that every item should consist of only one file. This gives a much better overview of all the files and by using this strategy, we can indicate differences between different versions in more detail by adding a second similar item.

Some item preferences we found are that every item should have a description that can be styled and that every item should have tags. These tags bring a great opportunity to allow users to search for similar items, based on the tags of all the items. These tags can consist of keywords about the work itself, even in great detail like material, technique, medium as well as inspiration keywords and even colors. If we store the time and date of an item, we could even query based on a time period.

A very important thing we found out was that we should provide some real-world example applications. Because the system will be quite abstract, we should create some example front-end applications that use the system, to demonstrate it's potential. We intend to provide some different examples to show the data, such as in a graph, a plain text page or even a PowerRail or XIMPEL[5] application.

We also found out that files should be protected from the outside world, to avoid having the files indexed to Internet search engines which reduces the server load.

Some things we noticed but would be very hard to implement would be the relations between items described with short text, which are very interesting but give a lot of extra programming work while this could also be achieved in the users application themselves. If multiple end-users would both be using the TOMS system we could create an API to send information over from one system to another, creating a cross-user portfolio for instance. This would require a lot of extra programming and would be a cool feature for a future version.

[4] PowerRail is a Papervision 3d and XML powered presenting tool, http://www.niftysystems.nl/powerrail

[5] The eXtensible Interactive Media Player for Entertainment and Learning offers the possibility to create interactive media applications. http://www.ximpel.net

Part II
Realizing the system

3 Terminology

To improve readability of the document, we define a few terms that will be used in the rest of this document to refer to certain things. By defining these terms, some of the database design requirements are already starting to become clear.

Item: a (creative) work made by someone or a company that is stored in the system

Author: the creator of a certain item. There can be multiple authors for a single item.

Project: grouping of items. in practice, items rarely come alone. Most often an item is a part of a group of items that may be better referred to as a project. A project may for example contain multiple items in the same series, or multiple items/objects that are part of the same contract with a customer. Of course sub-projects are also possible: these are subgroups of items.

Portfolio: the collection of works made by a single author, group of authors or company.

TOMS: the Object Management System back end, that stores items and authors and provides an API that allows portfolio visualization systems to retrieve portfolio items and their organization from the database.

Visualization: The visual representation of the items/work of a person or company in a characteristic way, using the content that is accessible through the calls to TOMS.

4 Representation possibilities

The system is designed to provide its content to the visualization application which can represent the content in its own way. The following examples give an idea of the possibilities an application developer has using our system:

- a concept graph of all “related items”, connected by the relation between types.
- a tagcloud based on all the tags within the system and their frequency.
- an hierarchical view of all items in plain text, as an addittion to a processor intensive application, for viewing the item on a less equipped device such as a mobile phone or netbook.
- an hierarchical view of all items by showing thumbnails of each item, showing the items author and its information when clicked.
- a flat view of items based on a certain author, shown as pictures in for instance a 3d carousel.
- a flat view of the items based on tags, for example clicking on a tag will show the items that have this tag.
- a flat view of the items based on type, for example showing a menu with all available types and showing all items after clicking on a certain type.
- a visual slideshow of all pictures and paintings in the system (e.g. PowerRail or XIMPEL).
- a visual slideshow showing all images in chronological slideshow.

5 Technology

This section describes why we chose which technologies to implement the opposed system.

5.1 MySQL

The first design choice we made was to use the opensource database software MySQL. As an alternative we could have used the enterprise MSSQL version, but due to our small budget and light usage, this is not worth the investment. The second alternative is the SQLite database, which is opensource as well and has the advantage of running locally in a file without needing a connection to a separate database server. We chose for the serverside MySQL program because of its speed, the easy replication, its ease of use because we have used this software multiple times before and because we have to use a webserver on the host for our application anyway.

5.2 PHP / AMFPHP

The second choice we made was to use the web language PHP for its usability, versatility, easy connection to flash, html, xml and other applications that are web based. Creating our back end system in PHP also has the advantage that it is available both to PHP web projects and Flash/Flex/ActionScript3 project using AMFPHP. AMFPHP is a PHP framework that allows for easy RPC connections to PHP frameworks from Flash for instance, which is our main target platform. Some alternatives to AMFPHP include SABRE and ZendAMF. We chose AMFPHP because we were already familiar with this framework and there was some high quality documentation that helped us along on http://www.gotoandlearn.com.

5.3 Flex / ActionScript3

The reason we used Flex and ActionScript3 as languages for the administration interface is that these are very powerfull and modern toolkits that have a lot of functionality built in out of the box. These built in elements such as itemlists, buttons, popups and windowing system are very visually pleasing and robust components to use in the application. Next to this, we also have some experience in Flex which will significantly speed up the development proces. An alternative would have been to build a standalone application in an application language such as .NET or Ruby, but these require a lot of setup time and have a steep learning curve. On top of that, these applications don't run on a webserver which is a very strong point of our object management system.

5.4 Flash

We chose Flash programmers / Flash artists as main target audience because Flash is de facto the internet standard for multimedia at the moment. Almost all of today's browsers are compatible with the Flash plugin and the penetration of actual installments of the Flashplugin is tremendous. Next to this we found that next to xml, it is very hard to include dynamic content inside a flash application. We want to eliminate editing xml as a method of organizing dynamic content and want to provide a more userfriendly and intuitive interface to do this.

6 Roadmap

- Brainstorm ideas (requirements engineering)
- SQL Database Design
- TOMS-Admin
- TOMSService
- Example representations

The roadmap we used to develop the system consisted of 5 stages. After deciding that we wanted to do this project, we started brainstorming ideas on what functionality we could fit into this project. This proces took a long while and can be summarized as an "informal evolving MoSCoW[6] list". During this period we studied several cases of artists that could be interested in using our system.

By spending a lot of time brainstorming and thinking about functionality we started the actual database design with the most important functionality prerequisites in mind which gave us a good starting and verification point. The database design was one of the most important parts of engineering within this project, for it has the biggest impact on how the data is organized and how it can be retrieved.

After we had the rough database design ready and implemented in SQL, we moved on to the flex content management system, which we later renamed to TOMS-Admin because it is essentially an administration tool to manage the content of the database. This tool loads and saves its data through the TOMS swc file. The TOMS-Admin application is just the administration program and is not a prerequisite for the custom flash application to work with the TOMS library. It is just a very handy way of adding, editing and deleting authors and items.

While implementing the items, authors and settings listing in the TOMS-Admin application we implemented the PHP querying functionality while we were programming. This may seem as a chaotic way of working and it may be so, but this gave us the advantage of having a working prototype at every stage of the project. If we would have postponed the PHP implementation we would find out later that the implementation we had in mind wouldn't work out because of functional limitations, or the other way around, restricting new ideas we got from the PHP point of view.

[6] MoSCoW list: a list of functional requirements categorized as Musthaves, Shouldhaves, Couldhaves and Wonthaves

7 SQL Database Design

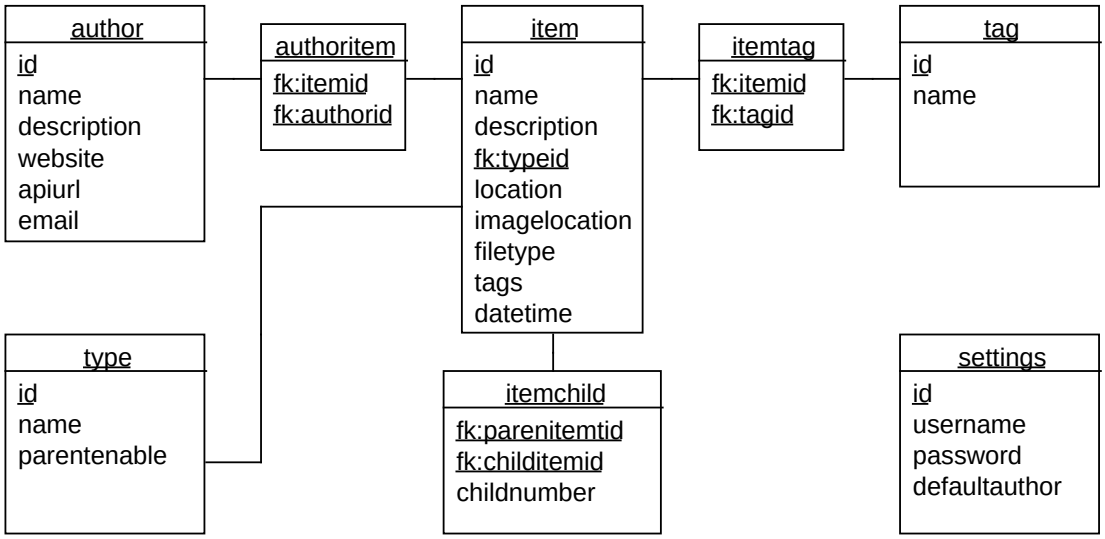

Figure 1: Database Design

Here follows a short description of all database tables and their fields.

7.1 Item

id contains the id of the item

name contains the name of the item

description contains the description of an item. This description can be formatted with BBCode[7] to enrich the text and to be able to include hyperlinks.

typeid contains a foreign key into the `Type` table, which contains a list of records that consist of an id and a name. In this way we can have a dynamic list of types and we ensure that the user can only assign an existing type to a specific item

location contains the location of the uploaded file, which is the id if the item appended with an underscore and the filename (for example `21_cheese.png` for item with id 21 and filename cheese.png).

imagelocation conatins the location of the uploade image or thumbnail, in the same formatting as the `location` but stored in a subfolder `images`.

filetype contains the extension of the uploaded file (e.g. `png` if cheese.png was uploaded)

tags contains a string of keywords seperated by spaces which represent the tags that the item was labeled with. This information is redundant because it is stored in the`tag` and `itemtag` as well. We left this field in to improve the speed of the application while loading and saving the items.

datetime contains the date and time of an item, which by default is set to the time the item was created. This field can be used to store the date the artwork was made, to enable the user to sort their objects based on time.

[7] see `http://en.wikipedia.org/wiki/BBCode.TOMSimplementsaslightlychangedsubsetofthestandardBBCode`

7.2 Itemchild

parentitemid is a foreign key to the the id of the item that we want to link to a certain childitem.

childitemid is a foreign key to the the id of the item that we want to link to a certain parentitem.

childnumber is an integer that is reserved for future use, to order the different children of a parent.

7.3 Author

id contains the id of the author.

name contains the name of the author.

description contains the description of an author.

website contains an url to the website of the author.

apiurl contains the url for the website api. This is a field we don't use but have reserved for future use, to for instance make cross-user systems's. This field could then be used to connect to when querying another authors database.

email contains the emailaddress of the author.

7.4 Authoritem

itemid is a foreign key to the the id of the item that we want to link to a certain author.

authorid is a foreign key to the id of the author that we want to link to the aforementioned item.

7.5 Tag

id contains the id of the tag.

name contains the name of the tag.

7.6 Itemtag

itemid is a foreign key to the the id of the item that we want to link to a certain tag.

tagid is a foreign key to the id of the tag that we want to link to the aforementioned item.

7.7 Type

id contains the id of the tag.

name contains the name of the tag.

parentenable contains an integer that is either 1 if this type is enabled to be a parent of other items or a 0 if it is not.

7.8 Settings

username contains the username that is used to login to the TOMS-Admin application.

password contains the sha1 hashed password that is used to login to the TOMS-Admin application.

defaultauthor is a foreign key to the id of the author that should be used as default in the TOMS-Admin application.

7.9 Choosing an database design for the tagging functionality

In order to get the tagging functionality to work smoothly, we researched different object tagging approaches, most notably:

- http://forge.mysql.com/wiki/TagSchema#Recommended_Architecture
- http://www.pui.ch/phred/archives/2005/04/tags-database-schemas.html
- http://laughingmeme.org/2005/04/07/in-lieu-of-the-promised-article-on-tags-and-sql/
- http://snippets.dzone.com/posts/show/34
- http://snippets.dzone.com/posts/show/35
- http://www.phpro.org/tutorials/Tagging-With-PHP-And-MySQL.html

In summary, there are three main tagging approaches: MySQLicious, Scuttle and Toxi. The first website above gives a very clear outline of the differences between the three approaches. We concluded that the Toxi solution is the most versatile, and the most useful for our systen.

8 Future improvements

- Slideshow implementation. A Slideshow would consist of the ordered list of its children, using the childnumber field in the itemchild table.
- Cross-user TOMS interaction. This function would allow two TOMS users to interact with both systems. For instance, if I had worked together with this other author on one project and you would like to see more work of this other author, you could request items from that author while these items reside on the remote authors TOMS system instead of ours. This would require a great deal of programming in authentication and messaging.
- Youtube or webpage link. Adding a field with a possible link and a type "Youtube" to the type table could enable users to upload references to youtube movies that could be incorporated in programs.
- Extend get functions. The getfunctions could be extended with numerous possibilities that are be useful for users.
- Item Rating. Items could be ranked within the flash application that is using TOMS to provide the programmer with functions such as getBestRated and such.
- Item Comments. It would be great if visitors would be able to comment upon items. This comment information could be used to relate items to each other. However it should be possible to moderate comments from the TOMS-Admin application.
- The Settings panel of the TOMS-Admin could be improved to allow for more settings to be changed, such as the size of the TOMS-Admin windows and other settings.
- Authentication for front-end function calls would make the system more robust. All functions that add, edit or remove data already require authentication.
- BBCode is currently only supported in the item description field. It would be very useful in the Author description field as well.