mediaTUM: digital collection management at TUM

Arne Seifert[1]

[1] Technische Universität München, Dept. of Computer Science, Boltzmannstr. 3, 85748 Garching, Germany, seifert@ub.tum.de.

Keywords
Virtual library, online repository, open-access, long-term preservation.

1. EXECUTIVE SUMMARY

In the actual services society everyone expects to acquire information very simple at any time. Behind this demand stands a quantity of infrastructure, which has to be provided on the one hand and on the other hand each part has to mesh correctly into another. At the Technische Universität München (TUM) the project IntegraTUM pursues exactly this goal - the creation of a user friendly and smooth infrastructure for information and communication (IuK) to the TUM.

In subprojects this slogan is to be converted by means of restructuring and re-centralization of individual services with simultaneous decentralized administration.

One project partner is the university library by providing a library portal, which summarizes existing search possibilities and integrates the university library better into the entire IuK infrastructure. This new service offers a central contact point for searching digital content such as theses or image collections. A substantial component is thereby the structure of a multimedia server, which opens archives for a broad number of users. For this purpose some existing products were sighted in order to be able to cover a broad spectrum of functionality. By collecting many demands (like e.g. long term preservation), we decided to build an own prototype, mediaTUM. Currently the project is out of the prototype state and the media-server is running since two years without problems. The high interest and the associated acceptance of the software is shown by many inquiries for the mechanism of test accounts and let become mediaTUM the model of success.

1.1. Background

In this paper some of the used concepts should be shown more in detail such as the main data structure, the search engine and the long-term preservation mechanisms.

In order to be able to deliver high-quality data, the multimedia server offers multiple possibilities to enrich digital objects with additional information during the ingest process. This information can be on the one hand metadata, which is maintained by flexibly adaptable masks with various types (for example lists or indices). On the other hand the inheritable rights management concept helps to secure objects, which enables the rights assignment by rules up to the physical object level (basing on username, group, date or ip).

The integrated configurable workflow engine opens the potential to make open-access possible for all users. In order to simplify maintenance, all functions provide web interfaces. Common interfaces like OAI are also available.

1.2. Conclusions

mediaTUM will be enhanced in case of new cogitations. During the last years the focus of some processes changes several times. This might sometimes to be connected with new experiences of the daily business. But sometimes the focus changes by outside influences coming along with changes in the administration process or workflow optimisation.

2. ARCHITECTURE OF MEDIATUM

Basic concept of the software is to support the user on every aspect of his daily work by accessing data. This requires a very flexible and configurable system that consists of different modules. Each module uses an API to communicate with the system core and each other. There can be made a primary partition into Frontend and Backend.

Generally all parts which are reachable by a normal user belong to the **Frontend**, either by using the common guest account or a special personalized one. Each of these parts will be delivered over the web server as html page. Adaptations in design and behaviour are possible through the flexible TAL-template engine to guarantee a seamless integration in existing environment. The **Backend** delivers all essential parts for the core system and builds the environment for each of the used components (for instance the complex internal data structure and the database layer).

Parts of the Frontend:

- **Retrieval:**
  This is the common interface for the normal user to browse through the content of collections or to search in different ways. Searching is one of the most important methods to gather information, so a simple and extended search functionality is included. For those who aren't looking for a special object there is also the browsing structure where objects are classified and can be structured by different categories.
- **Administration:**
  Administration area is a very special environment for user, rights and collection management. It's the place where all core system changes and configuration can be made. Collection management gives you the possibility to maintain datasets and metadata definitions for each object to be hold by mediaTUM. For each of these content types there are own editor masks to support the ingest process. You can only reach this section after authentication against the user management module.
- **Editor:**
  The user group of editors can made changes on the objects in this section of mediaTUM. These changes can be on the metadata and on the physical shape. Essential elements are the classification of objects and maintenance of metadata.

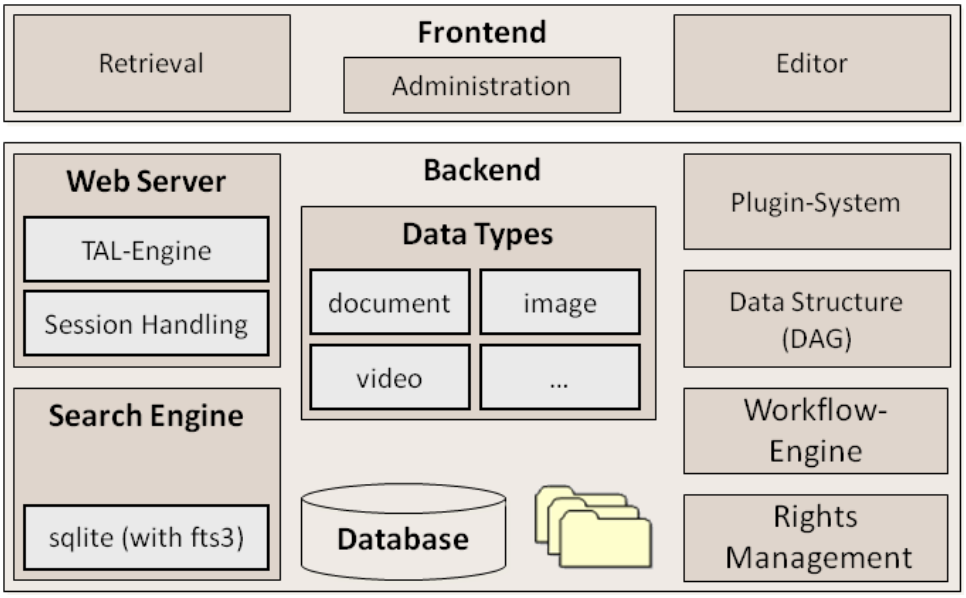


Figure 1: Architecture Overview

Parts of the Backend:

- **Web Server:**
  The web server delivers the formatted HTML content to the user of the software. This forms the service interface for user interaction.
- **Search Engine:**
  A powerful search engine guarantees the user to reach desired information in effective way. This goal is achieved by the algorithm of MG [9]. For python there will be used the wrapper magpy [5]. Currently there are still tests with other software.
- **Database:**
  Object information is stored in a relational database. Connectors for MySQL [6] and SQLite [7] are included at the moment. By the enclosure of the used SQL statements nearly every other database could be used, only the connector has to be adapted.
- **Data Structure:**
  The core of mediaTUM is the internal data structure for objects. Fundamentally it concerns thereby a DAG, a special tree like shape which is constructed out of the database information.
- **Workflow Engine:**
  In some cases a flexible workflow management is requested. MediaTUM provides a configurable interface that lets you build your own workflow out of simple extendable operation steps.
- **Data Types:**
  Managed objects are classified by its data type. Basic types like document, image or video are integrated and can be augmented by special types through the delivered interface structure.

2.1. Data Structure

Basic concept of the data structure is to ensure a flexible way to manage objects. During the ingest process of data and files there is no clear defined set of information to be hold inside the object. For that reason an extensible structure has to guarantee a high level of flexibility. This request is realized by the separation of meta information and object information. Object information is summarized in a small set of predefined attributes that are stringent necessary to manage the object as content by mediaTUM. However, meta data are a flexible set of data which are configurable by the administrator of the system.

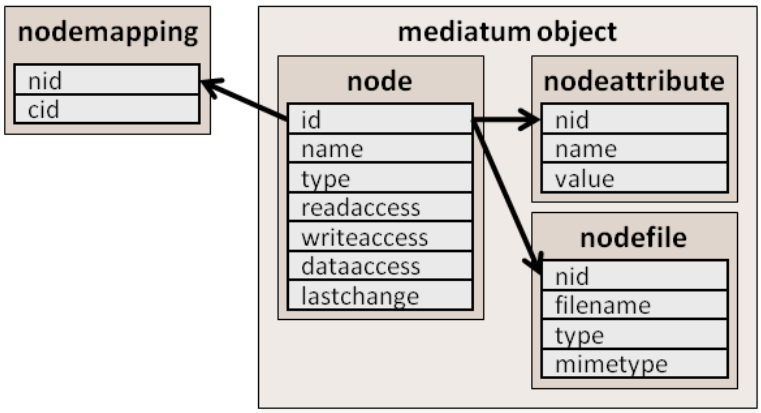


Figure 2: database schema

This separation is carried on till the database layer. Meta attributes are kept in an own relation (nodeattribute) separated from the system object information (node). Digital representation of each object can be for instance the text as pdf, full text extract and thumbnails. For other object types such as images the complete text section is missing. Information about the physical occurrence is stored in a separate relation (nodefile). The physical object is stored in the file system. Each object has a unique identifier id that can be used for object delivery later on.

Objects are organized among themselves in parent-child relations (nodemapping). This is a very flexible way to map hierarchical structures, because each mediaTUM object can occur in different

positions inside the system and is stored only once. Also caching mechanism can be used easily to enhance performance. Not only the content is stored in this *directed acyclic graph* (DAG), nearly all configuration and preferences exist in this way. To identify the functionality of given node the type attribute of a node is been used. With this type the correct object oriented class will be instantiated and delivers all special methods which are characteristic for this object.

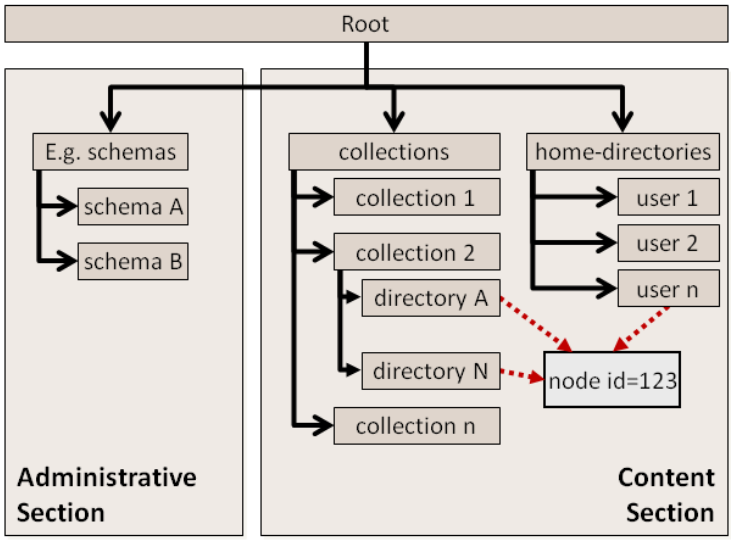


Figure 3: object structure

Currently there are the basic types for content (text, document and video) implemented in that way. But if there are other methods needed they can be integrated by this type of plug-in interface indentified by the type. Special nodes for maintenance and basic administration are stored in an own branch of the DAG.

As shown in Figure 3 the node with id=123 is used in *directory A*, *directory N* and in *user n* directory. To change the position there are only changes in the *nodemapping* table necessary. If there is no reference left on the object you can decide whether to delete the object or to store it in a special user folder.

2.2. Data Representation

The whole system is realized as object oriented software written with python. Each object class has to deliver some main methods for basic functionality for ingest, presentation and internal process. For that reason the behaviour of objects varies in different contexts. If we look at a text document for example:

- During the **ingest** process the full text has to be extracted to guarantee the search ability. This may cause the problem that there are a lot of different ways to produce pdf documents. In some cases there is no possibility to see the single character, only images are included. For those objects separate mechanisms are needed to extract text information (e.g. OCR). Additionally there are a lot of technical meta information which can be read out of the document such as the creator or the creating software.
- Later on there is the need of a symbolic representation for result listing. A nice feature is to have the first page out of the document as image. These **thumbnails** will be created during the ingest process, too.
- At the **frontend** each object has to deliver some different kind of representations. For preview there is a small view with only some major information of the object and a thumbnail if favoured. In additional there is a detailed view with all the meta information and the possibility to open the full text of the document. In some cases there are special viewers needed to provide the “original” object through the internet browser. Currently there are integrated viewers for images and videos. Documents can be delivered by the Adobe Reader.

The possibility to define standardized views on mediaTUM objects simplifies the presentation of content in an easy way. Each object “knows” needed information and can deliver them to the frontend. The template engine takes this data and format them equivalent to get a standardized view on the data.

In Figure 4 you can see two different contexts of the same mediaTUM object. It’s a text document with descriptive and technical meta information. In each case there will be a configurable selection out of the existing meta information used to display. In case of the close space only in the detailed view labels will be shown. Preview mode is normally used to visualize a list of matches or child

elements of a category. So only some major attributes will be shown. But you can define major attributes in the administration area to be completely free in your presentation. Additional templates of each view can be defined as favoured.

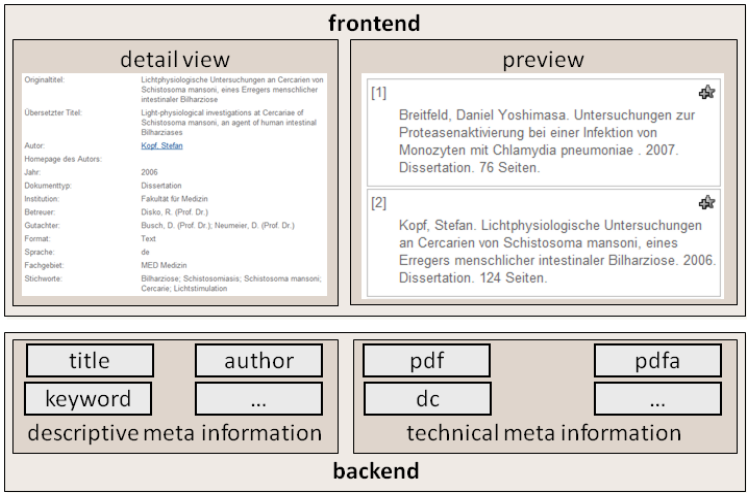


Figure 4: object in different contexts

3. SEARCH ENGINE

One of the major components of an online repository might be the powerful search engine. MediaTUM was designed with a special version of mg, managing gigabytes. This engine features fast full text search and indexing. Important is the possibility to use Boolean queries to search in index and full text values.

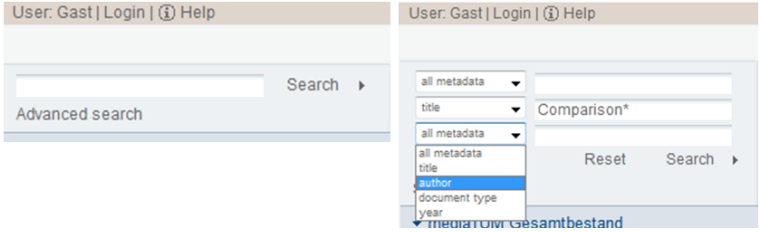
But there is one big disadvantage of this used algorithm: you have to re-index complete content at once and are not able to refresh only parts. For that reason the used index will only be updated once a day at specified time. Currently this behaviour seems to be more and more hindering. There are still tests with other search engines and algorithms that can re-index the content just-in-time.

The final decision was the usage of SQLite with the special module fts3. Fts3 can manage full texts and other textual information by a pre-process and delivers special query functionality to find text inside datasets with a lot of options.

3.1. Different Search Possibilities

The search functionality is positioned deep in the system and can be used on different areas in different ways. For the common user there are two positions on the normal frontend with simple and extended search. Simple search gives the ability to search for a simple word, phrase or truncated word. This search runs over all information managed by mediaTUM and delivers a list of hits sorted by collection.

To search more in detail you can use the extended search capability. This search can be defined by each meta schema and looks for the search term in combination with the given attribute inside objects. Three attributes can be combined in an extended search with “and”. So you have the possibility of a precision search through the content.



Editor users have in the frontend an additional position to search for more than three attributes connected with Boolean operations. This search lets you combine all searchable fields to find spelling mistakes or to correct information.

Figure 5: simple and extendend search at the frontend

3.2. Requirements

There are some requirements to the search engine and index:

- **Full Text:**
  Full text of documents has to be searchable not only in the simple way to find a single word. Phrases and truncation has to be found, too.
- **Attributes:**
  With mediaTUM you are able to define special attributes which has to be used in the extended search. These are normally pregnant attributes like author or title information of a document.
- **Actual Index:**
  After each change of data inside the system, index has to be kept actually.

Not only used content consists of typed nodes. All attributes of a meta schema are composed of nodes with a special type. This can be a simple text field, list fields with pre-defined values, index fields with lists of still existing values or some other typed fields. Each of these types can format internal values in a special way and has to be included in the search strategy.

But the major requirement is to have the possibility to combine all attributes by Boolean expressions. Only this possibility guarantees the possession to run complex queries.

3.3. Implementation

To comply with the requirements the implementation is divided into three different and independent parts. One part is the **simple search** over all object information, another handles the **full text** search with document data and the third one the **extended search** with the connection of search term and attribute name.

Each of these parts stores information in its own database table with different structure. Common is the index by mediaTUM id, the object type and the schema.

Simple Search:
The table with simple search entries is collecting all attribute values and stores them together in a database field. Search queries are looking for the search term in the value field and deliver a list of all matching object ids. Query example: *“full=’some search term’”*

Full Text Search:
In the full text table the textual content of documents is stored in an own value field. Generally there is only one entry per object. But in some cases the textual information are to long to put them together in one record. So there is an algorithm dividing the content into suitable pieces and stores them in more than one record. Queries behave like in the simple search mode and deliver id lists. Query example: *“fulltext=’some search term’”*

Extended Search:
This search mode is considerably more difficult and has to provide a set of rules for special cases inside queries. In one table there are stored all attribute names and the accordant mapping numbers depending on the schema name. The second table contains a set of fields for attributes (numbered 1 to 32). While building the index all searchable attribute values of each schema are stored in the same order in the table together with the object id. Search terms are containing additional information about the attribute to search in. This attribute name is mapped to the internal number and a query delivers all suitable ids in a return list. Query example: *“title=’first publication’ and author=’John Doe’ and publication-year >”2000-01-01T00:00:00””*

Search queries like shown above will be divided by a special Boolean parser and results in queries with only simple conditions:

“title = ’first publication’ and author = ’John Doe’ and publication-year > ”2000-01-01T00:00:00””

(1) “title = ’first publication’” AND
(2) “author = ’John Doe’” AND
(3) “publication-year > ”2000-01-01T00:00:00””

Each of these three queries delivers a list with matching ids. The complete result contains only ids from all of the three lists (intersection). Instead of the AND condition there can be an OR condition. In this case a union of result parts delivers the final result. Third operation is the NOT operator. This feature is a build-in functionality of the fts3 module. To negate a term, there has to be the prefix "-" ahead the term. Before delivering the result list to the frontend, rights management guaranties with a filter operation that only objects are delivered which are not shielded and invisibly for the current user starting the query.

3.4. Special Cases

Search queries can contain some special attribute names:

- **full**:
  used for simple search in all meta information
- **fulltext**:
  used for full text information of textual objects
- **schema**:
  to get all objects of a special schema. This is used for some interfaces (e.g. OAI)
- **objtype**:
  to get all objects of an given object type. For example all images
- **datehandler**:
  if search attribute is typed date, there is not only the equal operator possible, the comparison can also done by less or more.

Currently no more special cases have to be minded. In case there are new requirements the list of predefined special cases could be increased by additional keywords.

4. LONG-TERM PRESERVATION

One of the main requests to the system is the insurance of the long term preservation. Currently there are some initiatives working on this problem, resulting in guidelines how to ensure the further usage of digital content.

MediaTUM chases two different approaches which handles different parts of this problem. First part is to **backup stored information** and digital objects on secured disk space on independent media. Second part is to use **special object formats** suitable for the long-term preservation. The recording media for digital data deteriorate much more rapid than older once like paper. But the secure storage might be the easier piece of the problem.

4.1. Physical Deterioration

Physical deterioration is a very common problem not only for long-term preservation purposes. Nearly all systems using digital storage have to develop mechanisms to prevent deterioration. Unlike other storage media, digital storage normally losses the information completely and restoration might be more difficult or not applicable. For prevention there are some established proceedings like:

- **Redundant data storage:**
  Data will be stored at more than one hard drive (RAID). On mechanic fault or deterioration data can be reconstructed and stored on another mirrored hard drive.
- **Data backup:**
  Continuous backup of data on different storage media can prevent data loss because of their varying life-time.

4.2. Digital Obsolescence

The more serious challenge is to ensure the long-term access on the digital data. Currently there are a huge number of different data types for similar object types. Most of these types are proprietary and there is no insurance of accessibility in the future. For this lack of standard formats there are two strategies to prevent data-loss:

- **Emulation:**
  With emulation the complete environment will be emulated to simulate the media being in the creation environment. This can be very complex and expensive, because of the large number of different environments in their great number of versions.
- **Migration:**
  Transferring data in newer versions of system environments may help to prevent data loss. This strategy is only applicable if the number of different object types is straightforward.

4.3. Strategies of mediaTUM

Physical deterioration is managed by using redundant storage like RAID [3] for ensuring hardware faults. Daily backups are used to have a second fall-back layer. After the decision for the migration strategy there are some arrangements:

- MediaTUM will use some clear defined formats that are state-of-the-art concerning long-term preservation. This is the pdf/a format for documents, tiff for images and mpeg for videos.
- Predefined formats can keep their meta information inside the digital object. This helps to collect important information at only on place.

pdf/a and tiff can handle xml-attributes inside the file. For the semantic web [4] these functionality will be used to find the context of given objects. For the long-term preservation embedding of information can be used to store meta attributes of objects. The standardisation is nearly finished and RDF, resource descriptive framework, has been introduced [8].

The sample in Figure 6 shows the **subject-predicate-object** structure of an RDF term. Subject gives the namespace of the content (http://ns.adobe.com/pdf/1.3). Predicate can be used as attribute of the content (Producer and Keywords). And object holds the value of the predicate (Microsoft Office and Lorem ipsum). This functionality has the big advantage to deliver a powerful query language to be used with data objects. To get the information stored in a content object you have only to run a simple query.

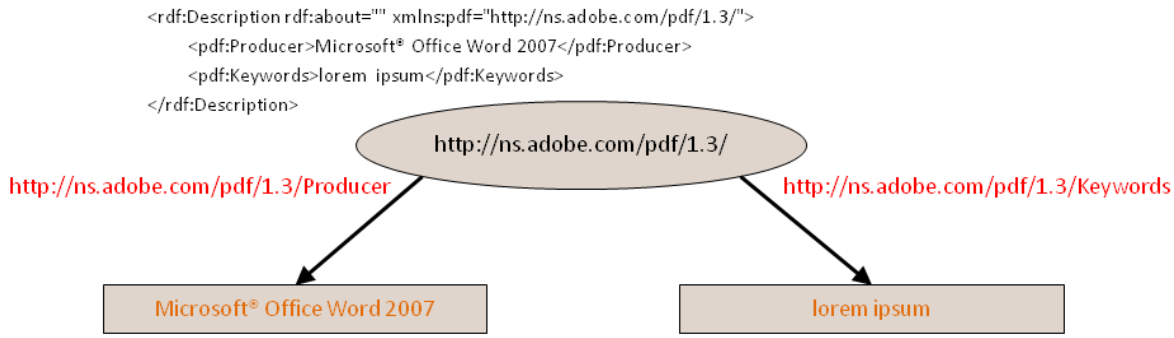


Figure 6: RDF sample

While the ingest process of a data object you can enrich the physical file with meta data attributes given by a predefined schema. This meta information can be stored directly into the file. Common file types used for long-term preservation for textual objects (pdf/a) and images (tiff) support the rdf feature inside special containers. This gives the ability to define following ingest process:

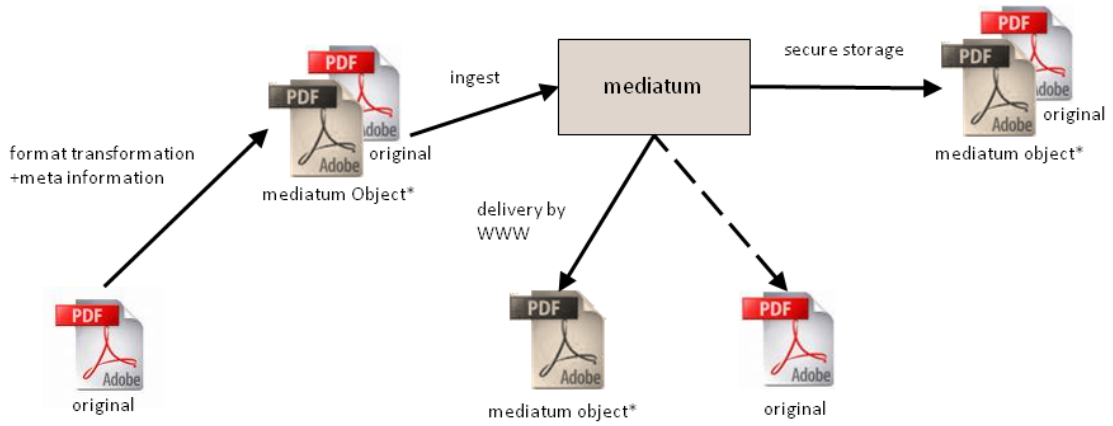


Figure 7: ingest process

Objects are managed by mediaTUM. Each textual object contains the *original* document and the *mediatum object* with included meta information in rdf. To secure the content all physical objects will be delivered to a preservation interface. In case of the TUM installation the Leibnitz Supercomputing Center (LRZ) assures the data access and backup. If someone requests data through the mediaTUM frontend the *mediatum object* will be delivered. Only in special cases the original physical file will be delivered. This behaviour can be transformed directly to image files in the tiff format. Additional functionality will be added by using the Tivoli Manager to backup system information.

5. CONCLUSION AND OUTLOOK

In this paper the approach for an online content repository designed by the TUM was presented. Only a small section of given functionality was described a little bit more in detail. Development of the mediaTUM software is still in progress and only a snapshot of the current state could be given. Some of the concepts were arisen out of the daily work. For example changes in the search engine becoming necessary out of the experiences in the live service during the last years.

Not all concepts are still integrated in mediaTUM. The area of long-term preservation is still in development. Currently only the conceptual work is done and interfaces are designed and ready to be implemented and integrated into the given environment. But this is not the only section in development. The software will be enhanced in case of new cogitations. During the last years the focus of some processes changes several times. This might sometimes to be connected with new experiences of the daily business. But sometimes the focus changes by outside influences coming along with changes in the administration process or workflow optimisation.

BASIC INFORMATION ABOUT MEDIATUM

Table 1: information about mediatum

programming language	Python
database support	MySQL, SQLite
platform	Linux, Windows
license	GNU General Public License
documentation	http://mediatum-pages.ub.tum.de
live system	http://mediatum2.ub.tum.de

6. REFERENCES

Borghoff, U. M., Röding, P., Scheffczyk, J., & Schmalhofer, F. (2005). Bewertung von Softwaresystemen zur Langzeitarchivierung digitaler Objekte. *ZfBB: Zeitschrift für Bibliothekswesen und Bibliographie 52. Jahrgang* .

Borghoff, U. M., Röding, P., Scheffczyk, J., & Schmitz, L. (2005). Langzeitarchivierung digitaler Dokumente: Technische Grundlagen, Initiativen und Projekte. *Auskunft: Zeitschrift für Bibliothel, Archiv und Information in Norddeutschland* .

Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., & Patterson, D. A. (1994, June). RAID: high-performance, reliable secondary storage. *ACM Computing Surveys (C SUR)* , pp. 145-185.

Consultative Committee for Space Data Systems. (Januar 2002). Reference Model for an Open Archival Information System (OAIS). USA.

Decker, S., Melnik, S., van Harmelen, F., Klein, M., Broekstra, J., Erdmann, M., et al. (2000). The Semantic Web: the role of XML and RDF. *Internet Computing, IEEE* , 63-73.

Kramm, M. (2005). *magpy*. Retrieved 2008, from magpy: http://athana.org/magpy/

MySQL. (n.d.). *MySQL: The world's most popular open source database*. Retrieved from http://mysql.com

SQLite. (n.d.). Retrieved from SQLite: http://www.sqlite.org

W3C, W. W. (2004, February 10). Retrieved 2008, from RDF/XML Syntax Specification: http://www.w3.org/TR/rdf-syntax-grammar/

Witten, I. H., Moffat, A., & Bell, T. C. (1999). *Managing Gigabytes: Compression and Indexing Documents and Images (second edition)*. San Francisco: Morgan Kaufmann Publishing.