THE PROBADO-FRAMEWORK: CONTENT-BASED QUERIES FOR NON-TEXTUAL DOCUMENTS

Rene Berndt[1]; Harald Krottmaier[1]; Sven Havemann[1]; Tobias Schreck[2].

[1]Computer Science, Graz University of Technology
Inffeldgasse 16c, 8010 Graz, Austria
e-mail: {h.krottmaier|s.havemann|r.berndt}@cgv.tugraz.at}
[2]Computer Science, Technische Universitaet Darmstadt
Fraunhoferstrasse 5, D-64283 Darmstadt, Germany
e-mail: {tobias.schreck@gris.informatik.tu-darmstadt.de}

Abstract

In this paper we describe the system architecture of PROBADO, a project funded by the German Research Foundation (DFG). Its main goal is to provide a general library infrastructure for dealing with non-textual documents, in particular for content-based searching. PROBADO provides an infrastructure that allows integrating existing data repositories and content-based search engines into one common framework. The system architecture has three layers interconnected by a service-oriented architecture (SOA) currently using SOAP 1.1 as the communication protocol. The layers are: [1] a front-end layer, responsible for providing the user interface [2], a core layer, responsible for scheduling requests from the interface to different repositories, and [3] a repository wrapper layer, responsible for enabling existing repositories and search engines to interface to the system. The functionality of each layer is described in detail. The general architecture is complemented by a brief introduction to the domain-dependent functionality currently provided.

Keywords: Generalized documents; multimedia retrieval; search engine architecture; web services.

1. Introduction

Textual documents are very well integrated in the workflow of today's libraries. Texts can be indexed automatically, metadata is almost automatically attached as well, and well-established user interfaces are available for retrieving and working with them. However, libraries have to deal more and more with multimedia documents: Digital images, 2D and 3D drawings, videos, animations, music in various formats, and even scanned 3D objects with millions of points. Multimedia data can be very domain dependent, such as weather data, physical measurements, or a collection of gears from mechanical engineering.

These types of digital content are not easily indexed, it is still difficult to retrieve such documents using search techniques such as query-by-example, and a variety of software tools are required to work with them.

It is the goal of the PROBADO project to make it as easy as possible for librarians and end-users to access and work with multimedia documents in real world digital libraries. To this end, PROBADO provides a framework for content-based indexing and retrieval of non-textual documents.

2. Background and Related Work

Building a Digital Library system is a complex process which requires addressing many conceptual and implementation aspects. On the conceptual level e.g., decisions have to be taken as to which kinds of document types and user interaction modalities to support. On the implementation level, many choices have to be made regarding the software technology, e.g. storage, description, transmission and presentation of content. Often, these aspects depend on each other on both the conceptual and implementation level, and they vary over the lifetime of the system. Digital Library systems are often confronted with conflicting requirements. For example, scalability with respect to the size of the repositories and number of users is required. On the other hand, the flexibility to integrate additional data types, retrieval methodologies, and content presentation modalities is required. These are some of the evaluation criteria to judge the effectiveness and efficiency of Digital Library systems. However, due to the diversity of document types and retrieval tasks to be supported, an objective comparison between such systems is difficult; very often the answer is simply “it depends”.

Certain results from theory and practice can be used to guide the implementation of a Digital Library. On the conceptual level, results from Software Design and Database Research indicate that a layered software design is advisable, at least providing a front-end and a back-end layer is reasonable to control complexity. The front-end layer can then be customized depending on the type of document to support. In the case of 3D object retrieval, several systems provide, for example, sketch-based user interfaces [1,2]. The back-end layer is responsible for evaluating the queries and for returning a ranked list of results to be presented in the frontend layer. In many systems an intermediate layer is also required for translating requests back and forth between the front- and back-end layers. The intermediate layer may provide important system services including indexing, user administration, or load balancing, or guarantee data consistency. One example of a Digital Library using a sophisticated middle layer is the DelosDLMS research prototype [3]. It employs the ISIS/OSIRIS middleware, providing monitoring of distributed processes and load balancing aspects. Modular architectures such as aDORe define conceptual reference architectures and sets of protocols by which system modules can communicate with each other [4].

Several popular Digital Library reference implementations exist which could serve as the basis for building customized Digital Library systems in practice. One often heard of alternative is FEDORA [5], an open-source system initially developed at Cornell University that is used as a document server in many research libraries. It follows conceptually the METS [15] approach and can specify certain complex document relationships using a template-based approach. DSpace [6] is another popular open source system developed by MIT and Hewlett-Packard. Further systems include the Greenstone digital library [7] developed at the University of Waikato, and OpenDLIB [8]. These systems have in common the fact that they are good starting points to set up a digital document server with support for OAI-oriented metadata. However, neither of these systems includes any specific support for dealing with non-textual documents, let alone content-based indexing and retrieval.

3. Requirements and Assumptions

The first assumption is that the document repositories belong to stake holders, and not to PROBADO. The owners of multimedia libraries will want to keep their data, and they typically only want make them searchable through PROBADO. So a content-based search in PROBADO yields as search result a set of hyperlinks of documents that match, but the target of the hyperlink belongs to the institution running the repository. The second assumption is that there may be more than one search engine per media type. A content-based search engine in principle receives as query the same media type as it looks for: A search engine for 2D drawings gets a drawing to search for. However, the content of the drawing may in fact be described in various ways: Line density, detail level, color histogram to look for, shapes to look for, or for the similarity to a blurred low-res bitmap that is specified in the search.

The third assumption is that it is possible to have more than one user front-end per search engine. So-called ***combined queries*** contain query data which go to more than one search engine; and ***cross-media queries*** even involve a query over more than one media type. An example is the search for a specific *musical box* (3D-object) that plays a certain type of *melody* (music).

The multimedia content PROBADO has to deal with is very heterogeneous, which is a great challenge for the system architecture. Multimedia indexing and retrieval is fundamentally different from text indexing and retrieval: We anticipate that there will always be new multimedia types, search engines, and repositories to add, the reasons are:

- **Media type dependency**: Surprisingly, the set of digital media types still keeps growing, from classical images, video, music, over 2D vector drawings and 3D meshes to animations (Flash) to RSS and Twitter. Also temperature measurements or deformable body simulations can be considered multimedia data.
- **Domain dependency**: Even within the same media type, different search engines (back-end) and query interfaces (front-end) may be necessary: Comic strips and architectural drawings are both 2D vector data. Buildings and scanned seashells are both represented as triangle meshes. Each of them, however, needs its own approach.

In order to make adding a new search engine or query front-end to the system as simple as possible, PROBADO must require only a few basic things.

Every repository that is registered needs to deliver to PROBADO (in regular time intervals) for each document a reduced core set of Dublin Core metadata. This makes it in fact possible to specify the text search in these multimedia data as the main search type that seeks through all registered repositories. – Every repository that is registered may specify additional Dublin Core metadata, depending on the media type and search domain (***extended metadata set***). Typically, some of these data are derived directly from the document (derived metadata). – Every search engine that is registered with PROBADO must be able to “speak” the common PROBADO multimedia query language.

4. System Architecture

The proposed architecture of the PROBADO framework is divided into 3 layers, see Figure 1. Communication between the layers is implemented using web services. Therefore, it will be possible to expand the functionality as long as the PROBADO protocol is implemented.

4.1. Layer 1 – Front-end

The front-end (Layer 1) is divided into two parts. Both parts are capable of presenting result lists of user queries. The first part is a lightweight web interface for handling textual access methods for searching in the metadata. The second part consists of a set of domain specific query clients. Each domain can provide specialized user interfaces for formulating content-based queries. Examples are a ***query-by-humming*** interface for music retrieval or a complex ***sketch-based*** interface for searching in 3D-document collections. The domain specific query clients can either be accessed by a web browser (with suitable plugin) or by extend stand-alone software clients.

Search Interfaces integrated in Web Browser

The PROBADO web service is currently hosted on a MS-IIS 7.0 running on Windows Server 2008. Even with standard HTML/Javascript, the user has access to all available search options, the core-metadata queries as well as content-based queries handled by the domain specific query engines in the repository layer. The web interface can also be augmented with specialized user interfaces for non-textual queries that can be easily integrated with web pages. These interfaces are usually implemented as browser plugins, for example, a JAVA3D interface for graphical input of musical queries, or a Silverlight/Native plugin for interactively assembled 3D queries. In particular, the lack of support for hardware-accelerated 3D within browsers makes native plugins necessary, which are unfortunately platform dependent.

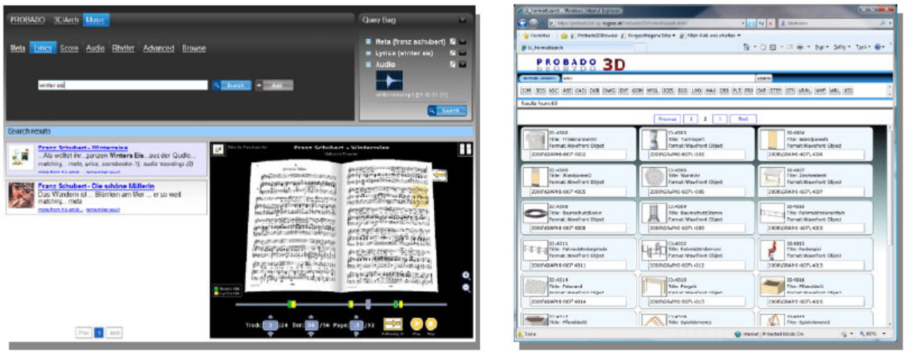

Figure 1: Query result sets for music and for 3D.

Search Interfaces integrated in 3rd party Software

Besides using web based clients it is also possible to use stand-alone software, or even to enhance 3rd-party software with the ability to access the PROBADO framework by using the product's plugin mechanism. A prototype is a script for the ***Google Sketchup*** modeling tool that allows using a created model as 3D query by sending it to the PROBADO content query (Figure 2).

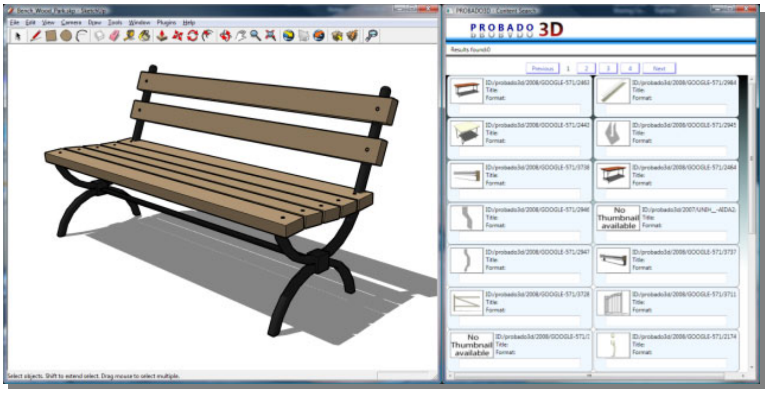

Figure 2: Google Sketchup used to formulate a 3D query.

4.2. Layer 2 – PROBADO Core

The core (Layer 2) is accessible through a well defined SOAP-API. In addition to the core metadata (see Metadata database), this layer also contains administrative data about the registered repositories (see Repository database), data on user sessions (such as query results and relevance feedback), and user profiles. The key part of this core layer is the ***dispatcher***, which either schedules queries to go directly to the core layers' metadata query engine (for efficiency, the core metadata are mirrored in the core layer of PROBADO), or sends them to the appropriate query engines located at the repositories. Result lists returned by the repositories are aggregated by the dispatcher, previews (e.g. thumbnails) are also integrated, and hyperlinks to special functionalities provided by the PROBADO framework (for example, an annotation service for non-textual documents) are created.

Repository database

This database stores all needed information about the registered repositories. The specific pieces of information, like title of the repository, description, accepted formats, etc. are shown in Table 1.

This database also keeps track of the current status of the repository (see Table 2), in particular, its online status. In case the repository is not available, a short message to inform the users (e.g. through the PROBADO portal web page) is stored. An optional time-span can specify update intervals to contact the server about metadata updates or system information requests. This should coordinate the load on the repository (e.g., to avoid sending requests while the repository database is running a backup).

Name	Description
DC_title	The title of the repository.
DC_identifier	The unique identifier of the repository. This is usually the URL of the WSDL description of the repository.
DC_description	A short description of the repository.
DC_publisher	Name of the publisher/organization that created/hosted the repository.
DC_language	All languages of the documents contained in the repository (ISO639-2).
DC_subject	Classification of the repository.
DC_format	MIME-types of the documents in the repository.
DC_type	Main subject area of the repository (e.g. "3D", "music",etc.)

Table 1 Repository data.

Name	Description
Available	Status flag if the repository is accessible or currently offline because of maintenance or problems.
Info	Description of the current status of the repository (e.g. "Repository down due to maintenance).
UpdateRange	Time-span in which meta-data update and system information requests should be sent.

Table 2 Repository status information.

Table 3 shows the entries describing each of the query engines of a repository. This information is used by the Dispatcher in order to route the search requests to the correct query engine, depending on type and domain.

Name	Description
Name	Name of the query engine (optional)
Retrievaltype	The type of queries accepted by this query engine (**Fulltext**, **Core_Metadata**, **Repository_Metadata**, **Content_based**)
Queryformat	Specifies the accepted MIME-types for the retrieval type "Content_based".

Table 3 QuerEngine information.

Metadata database

This central component of the core layer contains a subset of the original repository-specific metadata for all documents of all attached repositories. The schema of this database is mainly based on the Dublin Core (Table 4). Currently this database this database is mainly based on the Dublin Core (Table 4). Currently this database is hosted on a MS-SQLServer and the web services access the database using "LINQ-to-entities" (Language integrated query). The Entity Framework (EF) provides an object-relational mapping (ORM) to the database, so that the web service can operate on the conceptual schema of the database. The advantage of ORM is that the application – in our case the web services – can be abstracted from the physical schema, which makes this solution more independent from changes on the physical schema.

Name	Description
DC_title	Title of the document.
DC_identifier	Unique identifier (URN or DOI).
DC_description	A short textual description of the document.
DC_language	Language of the document in ISO 639-2 language-codes.
DC_date	One or more dates associated with the document, e.g. <DC_date category="Urauffuehrung in Prag">1787-10-29</DC_date> <DC_date category="Erstauffuehrung in Wien">1788-05-07</DC_date>
DC_creator	Creator/contributor. <DC_creator role="Komponist">Wolfgang Amadeus Mozart</DC_creator> <DC_creator role="Librettist">Lorenzo da Ponte</DC_creator>
DC_subject	Subjects for classifying the document.
DC_format	MIME type of the document.
DC_type	Subject area of the document, e.g. Music, 3D…
Repository_id	Id of the repository containing the document.

Table 4 Core metadata schema.

4.3. Layer 3 – Repository

The repository wrapper (Layer 3) connects the registered repositories to the core layer using a well-defined interface of SOAP-based web services. This layer acts as a façade and simplifies the access to the different repositories connected to the core layer. Each repository stores document collections of specific subject areas (such as 3D-documents or music) and the associated metadata. Access to each repository is provided by a set of query engines. Each of the connected query engines is registered with the core layer and provides a specific search functionality (like the query-by-humming for music collections). Using such a wrapper the repositories can run independently and still provide their full set of search features to the framework without losing control over the documents. If the search engine is located with the repository, the data in the repository do not even have to leave the repository in order to build a search index.

Each repository hosts its own metadata database. This database may also use a domain-dependent schema. All that is required for mirroring these entries in the core-database is a mapping to the Dublin Core subset, e.g. by providing a special database view.

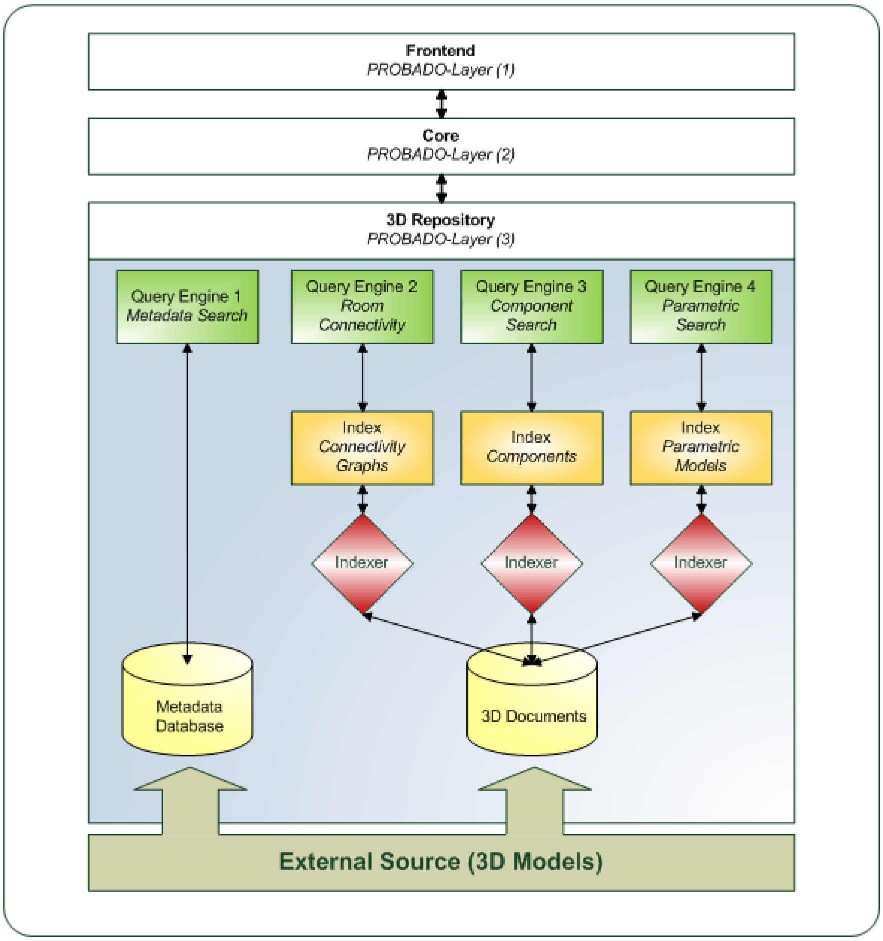
Frontend
PROBADO-Layer (1)
Core
PROBADO-Layer (2)
3D Repository
PROBADO-Layer (3)
Query Engine 1 Metadata Search
Query Engine 2 Room Connectivity
Query Engine 3 Component Search
Query Engine 4 Parametric Search
Index Connectivity Graphs
Index Components
Index Parametric Models
Indexer
Indexer
Indexer
Metadata Database
3D Documents
External Source (3D Models)

Figure 4: PROBADO 3D repository.

The various query engines are responsible for handling the SOAP requests from the core and returning the result set. There is no limitation on how many query engines a repository implements. Technically a query engine is implemented as web services. The following query engines are the most common ones and are implemented by the music and the 3D repository:

- **Repository metadata search**
- **Fulltext query** (e.g. searching in song text, or description or additional resources)
- **Content based query** (e.g. query-by-example).

Figure 4 shows the typical structure of a repository. The documents itself are not a direct part of this layer. They can be hosted by the content provider, in order to control access control, e.g. pay-per-view payment control stays in the responsibility of the content provider.

5. Communication

To establish an open architecture, PROBADO implements communication between the layers using well-defined SOAP-web services as described in Section 4. The first step for a document repository to connect to the core layer is by delivering specific information. A repository may have several query engines for separate types of query operations. For each of its query engines, a repository has to declare a particular query type and the accepted query format. Whereas the query format specifies what kind of data the engine accepts as an input (e.g. text strings, SQL statements, or MIDI files), the query type describes what kind of retrieval it provides (e.g. full-text retrieval, SQL-based database search, query-by-example for symbolic music). In a second step, each repository has to provide the set of core metadata derived form the internal metadata set, to the core layer.

A user can access the PROBADO framework using either the lightweight web interface or one of the domain specific clients provided by layer 1. Each of these clients directs its queries (containing the query data itself, a session ID, a desired range of query results, and optionally particular repository IDs) to the core layer.

In the core layer, each incoming request is processed by the dispatcher. Queries on the core metadata are passed to the core's local query engine. All other queries are distributed to either the user-specified repositories or to all connected query engines accepting the particular query format as an input (alternatively, all repositories providing a particular query type can be used). Upon receiving a query, each query engine calculates a ranked list of query results.

While collecting the results of the queried repositories, the dispatcher merges these to a single result list. This list constitutes the basis for the response to a user query. It contains specific details for each matching document, particularly a document ID, rank, title, description, accessibility information, context information, document type, and associated links. Furthermore, global information for each query like the total number of results, the range of results, the session ID, the query data itself, and IDs of involved repositories are stored.

When getting a response the front-end is responsible for presenting the result list in an appropriate format. Again there are the standard web interfaces for a basic display of the results (default) as well as the specialized client applications for creating complex representations (e.g. playback of audiovisual content). Regarding the result list a user has different options. Beside the details page available for each match, PROBADO provides a document preview.

ProbadoSearch.wsdl describes the the interface between the front-end (layer 1) and the core system (layer 2). The web service exports two methods for accessing the search functionality of the core:

- doProbadoContentSearch
- doProbadoMetadataSearch.

doProbadoContentSearch

Element	Description
Metadata	One or more key/value pairs containing the metadata. e.g. <Metadata key=“creator”>Mozart</Metadata> Each value has to be encoded in a single, if two or more values are required for the same key; e.g. <Metadata key=“creator”>Wolfgang Amadeus Mozart</ Metadata> <Metadata key=“creator”>Lorenzo da Ponte</Metadata>
CoreSearch	Specifies if the query shall be performed on the core metadata or will be used for a repository-specific query. Default is to search within the core metadata.

Table 5 Elements of the doProbadoMetadataSearch.

doProbadoContentSearch

This method performs a content-based query.

Element	Description
QueryData	Contains binary data used for the content-based query; the mimetype is specified by an attribute.
Parameter	One or more parameters as key/value pairs, e.g. <Parameter key=“engine”>sketch</Parameter> A fulltext search is performed by using <Parameter key=“Fulltext”>query</Parameter>

cont’d

cont'd

SortBy	The sort order of the results, e.g. <SortBy key="date">
RepositoryId	Specifies one or more repositories where the search will be performed. If no RepositoryId is given the request is sent to all registered repositories.
StartIndex	Starting index of the results to return (used for paging of results).
Count	The maximum number of results to return (used for paging of results).
SessionId	The session id of the core system.

Table 6 Elements of doProbadoContentSearch.

Result type

The result message contains besides "SearchQuery","StartIndex" and "SessionId" (which are just a copy of the request message), the following entries are returned:

Element	Description
TotalResultCount	Total number of results matching the query.
Count	Number of results returned.

Table 7 Common result elements.

For each search result item the following data entries are returned:

Element	Description
DocumentId	The core-specific document id.
Ranking	A value ranking
Title	The title of the document.
Accessible	Information how the document can be accessed (free, pay-per-view, etc.)
ContextInfo	Repository-depending information (e.g. short description, position within the document).
DocumentType	Mimetype des Dokuments.
LinkToDocument	The URL of the document.
LinkToPreview	The URL of the preview of the document (e.g. a 3D model with a low polygon count).
LinkToThumbnail	The URL of the thumbnail (e.g. a screenshot of a 3D model)
RepositoryId	Id of the repository.

Table 8 Elements of a single result item.

Request

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<rep:RepositoryContentSearch
xmlns:rep="http://www.probado.de/2008/01/31/RepositorySearch.xsd">
<rep:QueryData rep:mimetype="text/plain">
VGVzdGZpbGU=
</rep:QueryData>
<rep:Parameter rep:key=""/>
<rep:SortBy rep:key=""/>
<rep:StartIndex>0</rep:StartIndex>
<rep:Count>10</rep:Count>
<rep:SessionId>283E0C02C11BD8DB68CE8FFFD059DB64</rep:SessionId>
</rep:RepositoryContentSearch>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Body>
<ns1:RepositorySearchResult
xmlns:ns1="http://www.probado.de/2008/01/31/RepositorySearch.xsd">
<ns1:TotalResultsCount>158</ns1:TotalResultsCount>
<ns1:ResultElement>
<ns1:DocumentId>changeme:103</ns1:DocumentId>
<ns1:Ranking>-1.0</ns1:Ranking>
<ns1:Title ns1:category="unkown Category">JAVA Programming Guide -
Quick Reference</ns1:<ns1:Accessible>unkown</ns1:Accessible>
<ns1:ContextInfo>Java Programming Guide - Quick Reference Java Pro-
gramming Guide - Quick
<ns1:DocumentType>application/pdf</ns1:DocumentType>
<ns1:LinkToDocument>offis.core.probado.de/dokument?doc=changeme:103</
ns1:LinkToDocument>
<ns1:LinkToPreview>offis.core.probado.de/preview?doc=changeme:103</ns
1:LinkToPreview>
<ns1:LinkToThumbnail>offis.core.probado.de/thumbnail?doc=changeme:103
</ns1:LinkToThumbnail>
</ns1:ResultElement>
<ns1:ResultElement>
. . .
</ns1:ResultElement>
<ns1:SearchQuery>java</ns1:SearchQuery>
<ns1:StartIndex>0</ns1:StartIndex>
<ns1:Count>10</ns1:Count>
<ns1:RepositoryId>Probado-Core-OL</ns1:RepositoryId>
<ns1:SessionId>283E0C02C11BD8DB68CE8FFFD059DB64</ns1:SessionId>
</ns1:RepositorySearchResult>
</soapenv:Body>
</soapenv:Envelope>
```

Table 9 Example SOAP request and response.

The two SOAP messages described in **RepositorySearch.wsdl** "doRepositoryContentSearch" and "doRepositoryMetadataSearch" use the same structure as the doProbadoContentSearch/doProbadoMetadataSearch (with the exception of the field "RepositoryId"). An example SOAP request and response are shown in Table 9.

6. Conclusions and Future Work

The PROBADO system architecture as it was presented in this paper is only the first approximation to an integral framework for content-based queries in non-textual documents. Clearly, the priority was to design the system in a way that it imposes only minimal restrictions on the kinds of (i) ***repositories***, (ii) ***query engines***, and (iii) ***user interfaces*** that are to be registered. The goal was to reduce the hurdles for integrating with PROBADO as much as possible. Using standard web technologies, new repositories with different data types (e.g., photo or movie archives) can now easily be integrated. The PROBADO core layer allows associating flexibly query interfaces with repositories. One road of future work in PROBADO will be to extend the set of supported document types. However, eequally important at the current stage of the project is to advance from the restricted test operation to an ***enhanced test operation*** with first experiments on the integration of new external search services.

When the integration with the illustrated loose coupling works, future work will be to intensify the interaction between the different system components. The ***relevance feedback*** functionality, for example, requires some cross-media reasoning, to find a correlation between queries issued and results accepted. Another challenge is to allow active navigation through the result space, by continuously modifying a query, also allowing the possibility to ***backtrack***.

A more far-reaching goal would be a front-end technology that allows blending seamlessly from using one query engine to another. This could be important when, e.g., through relevance feedback, a user apparently changes the domain or the media type while navigating through the space of search results. This could be reflected in the user front-end: A search starting from 17th century church music could go over to church architecture and finally to the design of Gothic window tracery. In that case, one would expect a specialized graphical user interface to pop up that allows the interactive entrance, on a high level of abstraction, to the complex (and refineable) set of parameters of such a window.

Acknowledgments

The PROBADO project is funded by the German Research Foundation DFG. Its consortium consists of the University of Bonn, Technische Universitaet Darmstadt, the German National Library of Science and Technology in Hannover, and the Bavarian State Library in Munich. Cooperation partners are Graz University of Technology and Technische Universitaet Braunschweig. For further information, please refer to the PROBADO website at http://www.probado.de.

Notes and References

[1] FUNKHOUSER, T., MIN, P., KAZHDAN, M., CHEN, J., HALDERMAN, A., DOBKIN, D. and AND JACOBS, D. *A search engine for 3D models*, ACM Transactions on Graphics 22, 1, (2003), p.83-105.

[2] HOU, S., JIANTAO, P. and RAMANI, K. *Sketch-based 3D Engineering Part Class Browsing and Retrieval*. EuroGraphics Symposium Proceedings on Sketch-Based Interfaces & Modeling, (2006), p. 131-138.

[3] AGOSTI, M., BERRETTI, S., BRETTLECKER, G., DEL BIMBO, A., FERRO, N., FUHR, N., KEIM, D., KLAS, C.-P., LIDY, T., NORRIE, M., RANALDI, P. RAUBER, A., SCHEK, H.-J., SCHRECK, T., SCHULDT, H., SIGNER, B. and SPRINGMANN M. *DelosDLMS - the Integrated DELOS Digital Library Management System.* DELOS Conference 2007. Lecture Notes in Computer Science 4877, Springer 2007.

[4] HERBERT VAN DE SOMPEL, JEROEN BEKAERT, XIAOMING LIU, LUDA BALAKIREVA and THORSTEN SCHWANDER *aDORe: A Modular, Standards-Based Digital Object Repository*. The Computer Journal 2005 48(5):514-535.

[5] Fedora Commons (http://www.fedora-commons.org/).

[6] DSpace (http://www.dspace.org/).

[7] Greenstone (http://www.greenstone.org/).

[8] OpenDLib (http://opendlib.iei.pi.cnr.it/home.html).

[9] Open Archives Initiative.

[10] DIET, J. and KURTH, F. *The probado music repository at the bavarian state library.* Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR 2007). (2007) 501-504

[11] KROTTMAIER, H., KURTH, F., STEENWEG, T., APPELRATH, H.J. and FELLNER, D. *Probado - A generic repository integration framework.* In: Proc. European Conf. on Research and Advanced Technology for Digital Libraries, Springer-Verlag (2007) 518-521

[12] BLUEMEL, I., KROTTMAIER, H. and WESSEL, R. *The probado framework: A*

repository for architectural 3d-models. In: Browsing architecture. Metadata and beyond, Fraunhofer IRB Verlag (2008).

[13] WESSEL, R., BLUEMEL, I. and KLEIN, R. *The room connectivity graph: Shape retrieval in the architectural domain*. In: Proc. Int. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision. (2008).

[14] Generative Modeling Language (http://www.generative-modeling.org/).

[15] METS Metadata Encoding & Transmission Standard (http://www.loc.gov/ standards/mets/).

June 2009
Printed on demand
by *"Nuova Cultura"*
www.nuovacultura.it

Book orders: ordini@nuovacultura.it