METALICA: An Enhanced Meta Search Engine for Literature Catalogs

Bethina Schmitt *Andreas Schmidt*

Information Systems Group
Institute for Program Structures and Data Organization
Universität Karlsruhe (TH)
Am Fasanengarten 5, D-76128 Karlsruhe, Germany
E-mail: { schmitt | schmidt }@informatik.uni-karlsruhe.de

ABSTRACT
Today there is an increasing if not confusing number of services available for searching and acquiring literature. This makes it more and more difficult for a user to take advantage of these new services. Our goal is to offer user support by providing a meta engine which covers the different services and unifies their access. Additionally, we want to gain a number of synergy effects, enhancing traditional bibliographic information with a combination of complementary content and acquisition information. We also provide the means for a direct comparison of acquisition alternatives.

In this paper we present the design and implementation of our METALICA system. By employing a domain model and a global query language, heterogeneous services can be handled in a uniform way. Technical and syntactical homogenization is done by wrappers which consist of a connection control component and a syntax analysis component. Semantic homogenization is accomplished by mediators which contain a query translator and an attribute model translator. An integrator recognizes and unifies duplicates and offers additional operations for grouping and sorting, thereby supporting the user in the exploration of large result collections. The user interface can offer different views of the system´s functionality by utilizing a Model-View-Controller architecture.

KEYWORDS: digital library, user support, meta approach, distributed heterogeneous services, duplicate detection, domain model, flexible user interfaces

1 INTRODUCTION

The World-Wide Web offers an increasing number of services for search and acquisition of literature. The spectrum ranges from conventional library OPACs (online public access catalogs), across bibliographic databases, technical report servers, electronically available periodicals, full text archives, electronic document delivery services, to publisher's catalogs and online bookstores.

Even though this development fundamentally improves the availability of literature, it also places more excessive demands on the user. Especially in environments with a great need for information, like universities, a user is supposed to ask himself a number of questions: Where can I search at all? Where should I search? Should I consult my local library´s catalog first? Then I could check-out an interesting book, at least sooner or later. Should I search an online bookstore, which might hold additional information for a document? Then I could determine its relevance better, preventing me from acquiring a useless book. Do I choose a text which is immediately available online? In that case, the collection to choose from would be smaller and less representative, but possibly more up-to-date.

All these degrees of freedom create an open market of services for searching and acquiring literature. Competition forces providers to continuously improve their services. For a user to actually profit from this prospering and versatile market, three conditions must be fulfilled:

- A user must *know* different available services. There is some support by marketplace providers which collect references to existing services in a central starting point. Link collections in the WWW serve the same purpose.
- A user must *evaluate* different services. In the case of search services he could initiate searches with different providers. Meta search engines support this approach in the WWW.
- A user must *compare* different services directly with each other in order to be able to make a decision. In the area of literature this task is yet unsupported.

In this paper we apply the idea of meta search engines to the area of services for the search and acquisition of literature. There are, however, significant differences between a meta search engine for the WWW and the literature domain, which makes a number of enhancements necessary, but also provides an opportunity for a number of new features. An example is support for the third condition, enabling the user to make direct comparisons between results of different services.

Section 2 explains how meta search engines operate in the WWW, as well as the necessary adjustments and enhancements for the area of literature. Section 3 outlines the general architecture and operation of our system. In section 4, we introduce our domain model and global query language. Sections 5 and 6 deal with the implementation of wrappers and mediators, which are required for homogenizing the underlying services. Section 7 is concerned with the integrator, which computes the final result collection, thereby eliminating duplicates and applying various structural operations. We support multiple user interfaces, two of which are presented in section 8. A list of related work appears in section 9. Section 10 concludes our paper with a summary and plans for future work.

2 ENHANCING THE META APPROACH

Numerous search engines exist in the WWW. However, each of them covers only a small, mostly disjunctive part of the web [1]. Combined, they could provide much better results. Meta search engines like MetaCrawler [2] or Highway61 [3] pursue this idea. They do not keep data by themselves, all underlying services remain autonomous. They do offer a uniform interface to the user, allowing automatic and parallel access to the different individual services. This enables the user to concentrate on *what* to search for, instead of forcing him to remember *where* or *how* [4]. Further improvement can be achieved by eliminating duplicates based on title or URL equality, by grouping domains, and removing invalid links.

Compared to WWW search engines, services for the search and acquisition of literature show the following characteristics:

- The set of documents available from different service providers overlap, e.g. the holdings of publishing houses, bookstores, and libraries.
- Services are heterogeneous, they vary in query expressiveness and the amount and style of available document information. Examples are simple keyword search, fielded search and results containing a table of contents, proposed reader target groups, or availability for loan.

These two points show both potential for synergy and challenges to be mastered when employing a meta approach.

We illustrate synergies arising from the meta approach with an example. For a specific book the meta engine can tap into different services, obtaining in turn table of contents, cover art, different reviews, local loan possibilities, purchase prices, delivery times, or versions available online. All these information actively support the user in evaluating the relevance of a book and show him the availability through different channels.

When we transfer the meta approach to our domain a number of challenges arise, most of which are due to the heterogeneity of the underlying services:

- *Queries*: Different services vary in their query possibilities, both in the set of searchable attributes and expressive power of the query language. Here, it is not acceptable to be limited to the smallest common denominator.

- *Common schema*: Conventional meta search engines neither require nor offer a common schema for representing their results. In order to semantically process results of different providers and present them to the user in a homogenized form we need to establish a common schema.

- *Complex Service Structures*: To receive complete information for a document from a service, it is essential to follow additional links. Usually, the top level result provides a list of abridged document references. A second level then presents all available information for a single document. For instance, to obtain loan information from libraries it is necessary to follow at least one additional link.

- *Structural Analysis*: Returned information for a document (e.g. bibliographic citation) cannot be simply passed through to the user. An internal structural and semantic analysis, according to the common schema, is needed for further post-processing.

- *Detection of Duplicates*: WWW meta search engines can eliminate duplicates on the base of URLs. For documents, not only is definition of equivalence more difficult, but detecting and unifying them also requires more efforts, depending on the chosen definition.

- *Result Visualization*:. Large result collections, as they are expected from meta search engines, must be appropriately visualized to be manageable by a user. This becomes even more important in our domain, where each result contains a wealth of information, composed of a large number of attributes.

- *Interactive Result Processing*: Further user-level support can be added by post-processing the result collection [7]. Interactive operations like grouping and sorting help in better understanding of the result´s structure.

We have some more and rather general requirements for the design and implementation of our system:

- *Extensibility & Flexibility*: The area of literature and document information systems is a very active field where rapid changes and advancements are to be expected. Therefore, our system must be extensible to be able to quickly integrate emerging services. It also has to be flexible in order to cope with frequently changing HTML interfaces.

- *Fast Response Times*: As with every interactive system, fast response times are imperative. Since we cannot influence response times of the underlying services, we should at least minimize delays caused by our system.

Additionally, we want to keep the user informed about progress by continuously updating the result collection as new information comes in. Thereby, our system can start displaying results on par with the fastest server, instead of being forced to wait for the slowest one.

- *Appropriate User Interface*: Since it is difficult to assess a user interface´s suitability in advance, we aim for a flexible and easily configurable user interface. Our approach is based on the Model-View-Controller paradigm [6,15], where a system´s functionality is decoupled from its screen presentation and reaction to user input.

3 GENERAL SYSTEM ARCHITECTURE

Before we discuss each component in detail, we give an overview of the architecture of the METALICA system. Our design incorporates design ideas from meta search engines [4] and the layered I³-Reference Architecture [8], see figure 1.

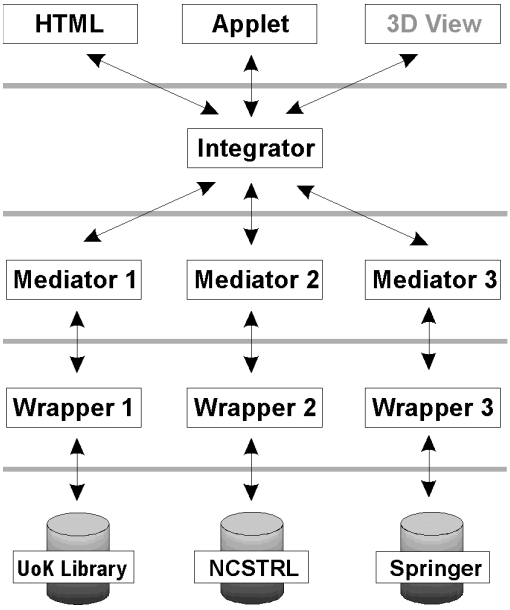

Figure 1: METALICA Architecture

In our approach, we are only concerned with bibliographic services offering an HTML interface. These services keep their autonomy and do not need to be modified in order to participate in our system. METALICA accesses the interface of each service, much like a user would do manually.

A user can interact with METALICA by choosing one from several available interfaces. Queries are transformed into a global query language and passed to the integrator, which in turn distributes it to the appropriate mediators. Each mediator is associated with a data source and translates queries into its specific language. The transformed request is sent to a wrapper which establishes a connection with the data source, executes the query, and receives the results. Based on the local schema of the data source, the wrapper performs a syntactical analysis of the results before they are returned to the mediator. The next step is a translation of the results, which are still in the local format of their data source, into the domain model, which is done by the mediator. The integrator continuously receives the results from each mediator, identifies document entities, and joins data records for identical entities. The resulting collection is presented to the user in the style of the interface he selected.

4 DOMAIN MODEL & GLOBAL QUERY LANGUAGE

For achieving an integration of heterogeneous services, our approach relies on the definition of a domain model in contrast to common strategies for schema collation, e.g., [9]. A domain model covers relevant aspects within the field of application independent of structures and attributes of actual services. Therefore, a domain model is robust and remains stable even if actual services undergo changes in attributes or structures. A domain model is manually created by experts. Additionally, transformation rules for mapping attributes of services to the domain model must be defined. Even though this approach is quite expensive, it nevertheless is the only way to both obtain exactly relevant information for a document and to present to the user acquisition information that is directly comparable.

Essentially, document information falls into three categories: bibliographic information, content information, and acquisition information. In the process of creating the domain model we developed a formal representation for each of these categories. As a prerequisite, we include information that is currently available from different services and makes sense to a user, as well as information that is needed internally for our system, e.g. for detecting duplicates or visualization.

4.1 Bibliographic Information

Within the area of bibliographic citations a general understanding has been reached on what kind of information is essential for describing a document to a user. For this purpose USMARC [10], defining several hundred attributes, is too fragmented, whereas Dublin Core [11] is too unspecific. Our model employs about the same level of detail as bibliographic entries in the German RAK-WB[1] (rules for alphabetical cataloging for scientific libraries, [12]), which essentially corresponds to the AACR (Anglo-American Cataloguing Rules [13]).

Our domain model contains the usual bibliographic features, such as `title`, `author[]`[2], `publisher`, `publishing date`, `edition`, `ISBN`, `ISSN`, `key word[]`[3], `subject heading[]`[4], and `classification[classification scheme`[5]`, classification code]`. Other attributes are `number_of_pages`, `language`, and `type_of_publication`, similar to BibTeX entries. These attributes are needed to allow for enhanced queries, like "German text books", performing operations on a result

---

[1] Regeln für die Alphabetische Katalogisierung für Wissenschaftliche Bibliotheken
[2] [ ] symbolizes set-valued attributes
[3] uncontrolled vocabulary
[4] controlled vocabulary
[5] e.g. {(ACM,B.C.3), (DDC,005.019)}

collection, like grouping by type of publication or language, or different visualizations, like a 3D view rendering number of pages or type of publication.

4.2 Content Information

Recently, publishing houses and online bookstores have been adding more and more varied information to their documents. These pieces of information are very helpful for a user in evaluating a document´s relevance or usefulness. Consequently, our domain model contains attributes for: `text_description[]`[6], `table_of_contents[]`[7], `cover_art`[8], `full_text[]`, `target_group[]`.

4.3 Acquisition Information

The handling of acquisition information is a very young field and no common standard has yet been established. Therefore, we designed this part of our domain model from scratch, cooperating with experts from the library at the University of Karlsruhe.

A user currently has a number of different acquisition alternatives. Examples are local libraries, online orders from publishers or bookstores, as well as full text archives and electronic document delivery services. In our domain model we provide the following representation:

`provider[ ]`[9]	of document references
`supplier[ ]`	of real documents
`name`	
`opening_hours`	
`reference_id`	e.g. shelf mark for libraries
`holdings[ ]`	only for periodicals
`from`	
`to`	
`num`	
`delivery[ ]`	
`format`	online, email, loan, read, ...
`time`	obtainability in hours/days
`cost`	
`location`	URL, reading room, check-out desk

A certain document can be available from different suppliers, possibly in different formats. If the book is supplied by a library it could be either immediately available for loan from one branch or after a two-week waiting period from another. It could also be available from a local bookstore for a fixed price or as an electronic version, to be downloaded from the given URL. Additionally, it might be ordered from an online bookstore where it would be delivered by mail within a certain timeframe for a certain cost.

4.4 Global Query Language

The integrator component of our system expects all queries to be expressed in terms of the global query language. The actual query formulation at the user-level interface is thereby independent from its internal representation. This allows for experiments with different query interfaces without having to change the global query language or internal system components. Figure 6 shows one of the currently available query interfaces.

METALICA is not restricted to queries based on bibliographic or content search criteria. Rather, all attributes of the domain model can be incorporated into a query. Additionally, a typical query contains the list of providers to be searched, required output fields, grouping and sorting of the result collection, and limitations concerning response times or result sizes. Finally, the preferred display format can be chosen, see figures 7 and 8 for a plain text and Hi-Cites view.

The corresponding internal representation is similar to SQL:

```
SELECT    list of output fields
FROM      list of sources
WHERE     search conditions
GROUP BY  grouping/sorting criteria
OPTION    performance restrictions
VIEW AS   view name
```

A search job formulated in such a way is subsequently passed from the integrator to the mediators, which in turn interact with the wrappers in a single or a number of queries in order to obtain the requested result.

5 WRAPPERS

The wrapper layer achieves a technical homogenization of the services. Each wrapper is responsible for transmitting a given query to its data source, receiving the resulting document, and extracting data fields needed by its mediator. Therefore, a wrapper contains two principal components: a connection control and a syntax analysis component.

The connection control component builds an HTTP request for a given query, depending on the data source´s access method (GET/POST). It then establishes a connection, receives the resulting document, and is also responsible for handling access errors. The result can optionally be transcribed and translated to obtain a pure UniCode string representation, which is internally used by Java. The transcription thereby converts different character sets (e.g. ISO Latin 1) and the translation replaces strings with other strings (e.g. “&szlig” with “ß”). Consequently, each connection control component can be parameterized with a transcription and translation table.

The syntax analysis component parses the resulting document, obtaining a representation in a simple object model (OEM, [14]). The OEM supports both flat structures for representing attribute-value pairs and hierarchies for representing syntax trees. We employ the strategy design pattern [15] which allows for a flexible exchange of syntax analysis methods.

Conventional HTML wrappers are not concerned with syntax analysis since they only deal with data sources which already tag each semantic unit within their results. But we

---

6 abstract, blurb, excerpt, review, ...
7 often available in several formats
8 URL
9 provider of a search service which references the specified document

do not wish to restrict the choice of data sources, therefore it is not sufficient to rely on simple HTML/XML parsers. We have developed a new method for syntax analysis which we call *hierarchical regular expression parsing*. The implementation is based on a special class library [16], which supports the specification of regular expressions.

At this point, we omit a formal definition of hierarchical regular expression parsing. Instead, we show the parser grammar as it is used within the wrapper for the NCSTRL service [17]. Figure 2 shows a top level result list and figure 3 the corresponding part of the specification file.

Within our implementation, all information needed for parsing the result format of a source are stored in separate specification files. This solution gives us the needed extensibility and flexibility. Modifications of the result format can be dynamically handled through adjustment of the hierarchical regular expression, without requiring a re-compilation of the wrapper. A new data source can easily be integrated into our system by creating an appropriate specification file.

6 MEDIATORS

Mediators translate queries expressed in the global query language into one or a series of wrapper requests. The returned results, which have already been syntactically analyzed by the wrapper, are transformed into attributes of the domain model. Mediators are equipped with knowledge about the global query language and the domain model, as well as the query syntax and result structure of its assigned wrapper. According to these tasks, a mediator contains two components: a query translator for the direction from mediator to wrapper and an attribute model translator for the reverse direction.

The query translator receives a request in the global query language and translates it into a query execution tree. Such a tree consists of nodes, representing either set operations, filter operations, or expansion operations, and of leaves, denoting queries which are passed to the wrapper, see figure 4. Set operations are needed to address sources which allow only limited or no Boolean queries at all, like some publisher catalogs. Filter nodes deal with query conditions that are not supported in the query language of a given source, e.g. `language=german`. Expansion nodes are important for triggering follow-up queries which collect information (e.g. loan information) that can only be obtained by following additional links. Each tree node contains an operation which is performed on the results of its children, which in turn constitutes the result of the node. Finally, a tree is processed if all nodes have been successfully executed, starting with the leaves and continuing bottom-up to the root.

So far, we implemented a query translator which is configurable with a mapping table for attribute names and includes set operators for union, intersection, and difference.

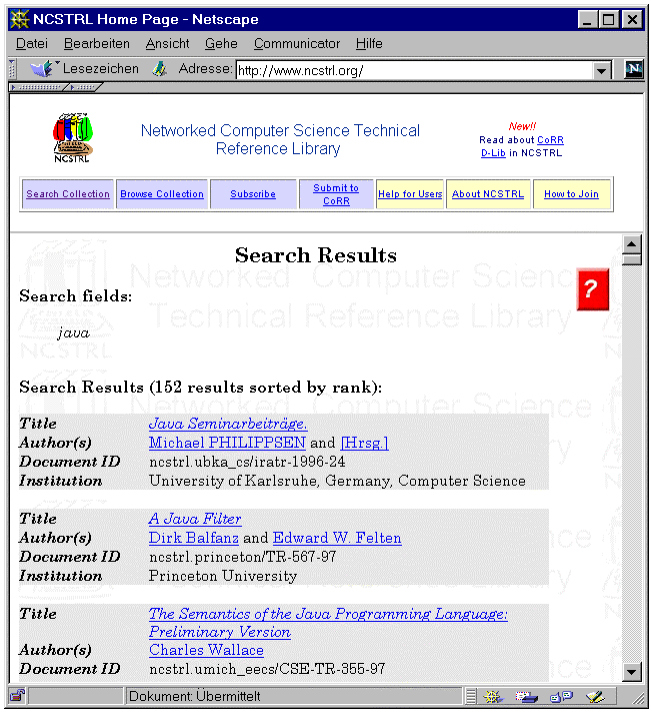

Figure 2: NCSTRL Result List

```
Result      <H3>Search Results
            \(@HitCount:\d+@.*?</H3>
            @DocList@<P>
DocList     SPLIT[POSTFIX] "<p>" @Document@
Document    <tr>
            title
            <A HREF=\"@TitleURL@\">
            <i>\s*@Titel@\s*</i>
            <tr>
            Author\(s\)
            <td [^>]*>\s*@Authors@\s*</td>
            <tr>
            Document ID
            <td [^>]*>\s*@ID@\s*</td>
            <tr>
            Institution
            <td [^>]*>\s*@Institution@</td>
Authors     SPLIT[INFIX] "(?: and )|(?:,)"
               @AExpr@
AExpr       <A HREF=\"@AuthorURL@\">
            @Author@</A>
Author      @FirstNames:.*@ @LastName@$
FirstNames  SPLIT[POSTFIX] " " @FirstName@
```

Figure 3: NCSTRL Parsing Specification

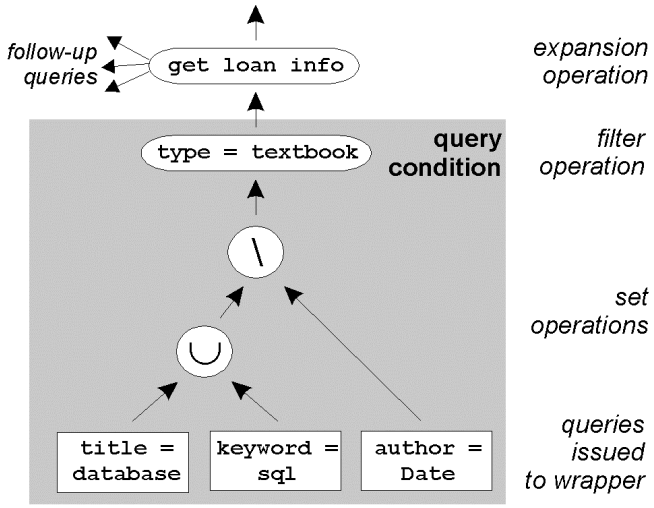

Figure 4: Query Execution Tree

The mediator´s second component, the attribute model translator, changes the structure of OEM objects returned by the wrapper. The received object contains values still formatted in the structure of their source. The mediator now rebuilds the object according to the structure of the domain model. In this transformation process, information can be discarded, completed, or converted, e.g. conversion of currencies or language codes.

```
"NCSTRL" -> $Catalogue
Result : Result
{
   _ : Service
   {
      HitCount -> Count
   }
   DocList : Documents
   {
      Document : Document
      {
         Authors : Authors
         {
            AExpr : _
            {
               Author -> Author
            }
         }
         TitleURL ->
            [encapsulate($Catalogue,
              "SingleDoc")] TitleQuery
         Titel -> MainTitle
         ID -> ID
      }
   }
}
```

Figure 5: Mediator Specification

Similar to our syntax analysis approach we developed a comprehensible specification language for mapping between different attribute models. The language offers a high-level abstraction for navigating an OEM-source-tree and composing the corresponding OEM-target-tree. Examples of operations that can be expressed in our language are renaming of attributes, transformation of attribute values, and the use of conditions and variables. Additionally, it is possible to traverse existing trees structures and create new ones. Again, we omit the formal language definition in favor of an example for an attribute model translator specification. In this example, which is shown in figure 5, the attribute model translator is responsible for translating NCSTRL´s results into the domain model. To remain flexible here as well, all required definitions are kept in separate specification files.

7 THE INTEGRATOR

Whereas wrappers and mediators homogenize the result collection of each data source the integrator joins these results together to the final result collection. Duplicates are eliminated, depending on a configurable equivalence relation. The remaining documents are then grouped and sorted into a nested list structure. Finally, the model component manages the result collection, thereby offering operations for the list structure and single documents.

The duplicate detection component identifies equivalent documents received from different services and collates them into a single equivalence class, which is then used for generating a representative document. For an implementation, two additional points have to be considered:

- Different definitions of document equivalence are possible and must be supported. Often, it makes sense to consider different editions of a book as equivalent. But in other situations exactly the distinction is crucial for a user, for example when he is only interested in the second edition.
- To be able to offer a fast and responsive user interface, we need to incrementally grow the final result collection. Therefore, the integrator must be able to continuously augment its result collection while new documents and their affiliated information come in.

We implemented a generic strategy for duplicate detection based on equivalence relations. The equivalence relation currently in use is the n-gram method described in [18]. After normalizing the `title` and `author[]` values, we employ tri-grams with a threshold value which linearly depends on the total number of occurring tri-grams in both character strings. Because the result collection grows incrementally, equivalence classes cannot be computed in a two or multi-level process, as it is usually done. Instead, each document is compared with all existing equivalence classes as it arrives. Thereby, it is either inserted into a suitable existing or a newly created class. If additional information arrive for a document its classification has to be re-checked.

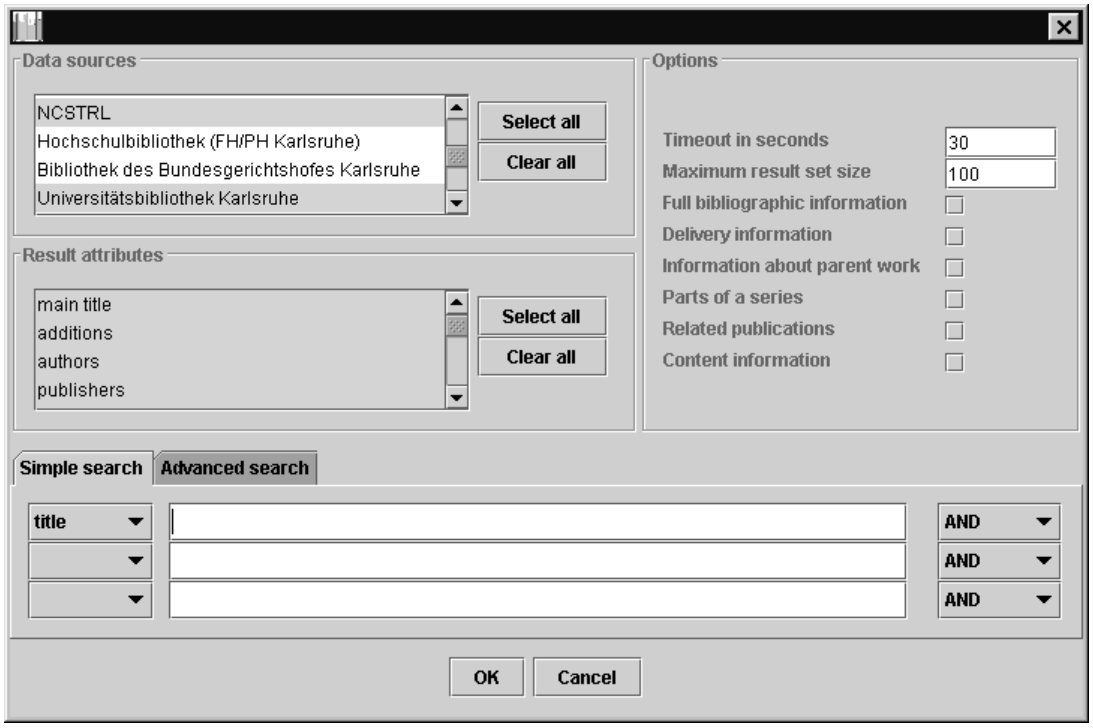

Figure 6: Query Interface

The grouping component is in design and function similar to the duplicate detection component. It additionally supports multi-level grouping of documents. It also manages the nested document list where each level has an assigned grouping criterion (e.g. year). Documents are grouped in sublists which are identified by a descriptor (e.g. `year=1998`). New documents are added to a group on the base of these descriptors.

Two generic grouping strategies have been implemented. Strategy 1 arranges documents with the same attribute value into the same group, thereby generating a disjunctive classification. Strategy 2 is suitable for set-valued attributes and groups documents where the intersection of their attribute values is not empty (e.g., one author in common).

The sorting component is called by the grouping component when a new document has been added, effectively sorting the specified group. If a new group has been created the list is sorted on that group´s level. Further work needs to be done on a multi-language support within the sorting component. So far, a language dependent sorting strategy (ascending, descending) for single-valued attributes has been implemented.

The model component maintains the final result collection for a query. It also offers a number of operations on the result collection, available for the other integrator components and the view components of the user interface. Namely, there are operations for:

- adding new and modifying existing documents,
- reading information for a document,
- reading and modifying the nested list structure,
- configuration of the duplicate detection, grouping, and sorting strategies,
- issuing follow-up queries, and
- notifying observers for view synchronization.

8 THE USER INTERFACE

The user interface interacts with the user, allows him to specify queries, and displays their results. The query interface design takes the recommendations of [19] into account, see figure 6. Data sources can be selected from a list holding all available services. Another list allows the user to choose attributes for display in the result documents. Currently, queries can be formulated in two ways: simple (fielded) and advanced queries. For the fielded search, the user selects search attributes, which can be combined with Boolean operators, and enters the search terms. The advanced search is available for experts who want to formulate unrestricted Boolean queries. A number of additional options control the result size, response time, and other parameters.

A search is explicitly started; we do not offer a designated result preview. However, since we do grow and display the result collection incrementally, a fast provider can serve as a preview function.

Results can be displayed in a standard HTML text view or in another view, which maps the result documents to a tree

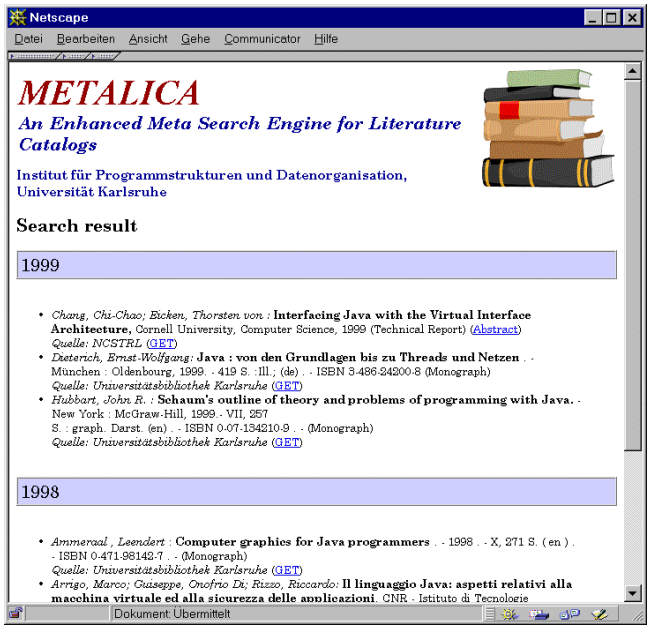

Figure 7: HTML-Text-View

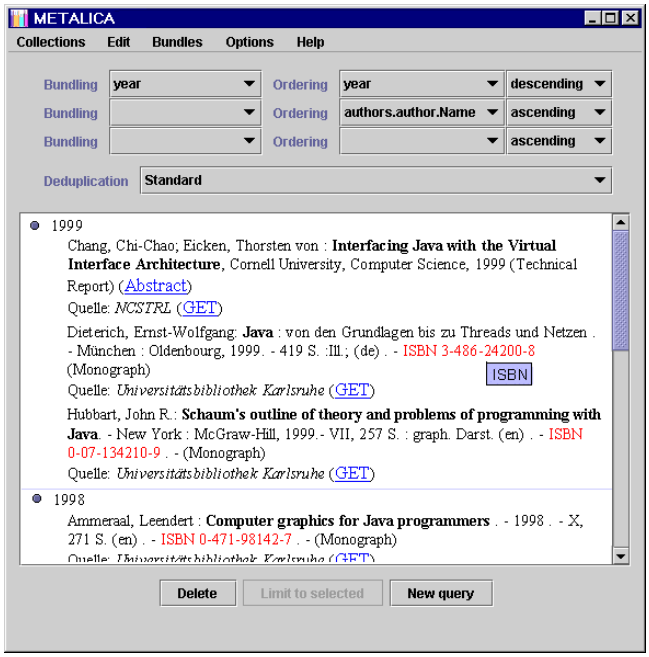

Figure 8: Tree-View with activated Hi-Cites

structure. Nodes and leaves correspond to document groups and single documents, respectively. This interface has been implemented as an applet with the Swing package, taking advantage of Java´s internationalization feature.

All tree leaves offer the Hi-Cite functionality [20]. These highlightable citations combine the advantages of a compact textual representation with the clear structure of a table representation. The Hi-Cite functionality is activated if the mouse cursor remains on top of an entry. As a result, the same attribute is highlighted in all the other documents. One benefit is the straight and easy comparison of document attributes, like cost or availability.

Figure 8 shows the tree view for a result collection, grouped in descending order by year and then in ascending order by the first author. Here, the Hi-Cite function has been activated for the ISBN attribute, highlighting the ISBN numbers of all documents in red. Additionally, a tooltip window reveals the name of the emphasized attribute.

A user can now rearrange the result collection by selecting different grouping and sorting criteria or applying different equivalence relations. In this case, the user interface calls the corresponding operation within the interface of the integrator´s model component. These operators then adjust the nested list structure maintained by the integrator. Finally, the model component notifies the affected observers, causing the views to be re-drawed.

9 RELATED WORK

The Karlsruher Virtueller Katalog (KVK, [21,22]) is a very successful meta engine for literature search. However, it can only offer a uniform presentation of author, title, and year information. Also, it neither offers duplicate detection nor post-processing operations. Results are only displayed in a standard text-based HTML interface. In order to obtain further bibliographic, content, or acquisition information a user has to manually follow many additional links, leading to different providers. There is no support either for acquiring documents or direct document comparison.

Medoc [23] is a meta search engine for different full-text providers within the area of computer science. It requires some coordination with the service providers. The document supply is rather small, but all of these documents can be obtained immediately, either free of charge or after buying rights for online reading or printing. Since the set of documents available from all providers is disjunctive, it neither requires integration nor special support for the comparison of acquisition alternatives.

DealPilot [25] is a shopping agent for online bookstores. Given a document, DealPilot starts to collect acquisition alternatives from at present 25 providers. The results are presented in a table with a row for each online bookstore and columns holding information about the prices of a book, possible discounts, delivery costs, and delivery times. DealPilot gives a user exceptional market control within the restricted area of online book shopping by providing him with a comprehensive survey of the market. Unfortunately, DealPilot does not support libraries, full text archives, and numerous other service providers, all of which are important for obtaining scientific literature and which should play a competitive role in the overall market.

The Stanford InfoBus [24] provides a set of models and protocols for accessing various kinds of information sources and services. Using this very generic architecture based on CORBA distributed object technology would cause much overhead. In contrast, our METALICA system is both lean and functional for practical use, especially tuned to the genre of literature services.

10 SUMMARY AND FUTURE WORK

Our METALICA system aims to assist a user in benefiting from the prospering online market of literature search and acquisition. By combining different services, we strike for a number of synergy effects, making way for advanced user guidance like a comprehensive, semantically homogenized comparison of acquisition information, obtained from a multitude of various service providers. METALICA applies the idea of meta search engines to the area of literature catalogs. We devised a number of enhancements that were needed to overcome heterogeneity and to enrich user-level support.

Heterogeneous services are integrated on the base of a domain model, which we designed to incorporate all facets of a document, including traditional bibliographic information, diverse content information, and information needed to acquire a physical or digital version of a document from a commercial or public entity. Our global query language decouples the user interface from the system´s core functionality, permitting us to modify them independently; an important aspect for further experiments, e.g., in the area of novel query interfaces.

Wrappers and mediators deal with the syntactical and semantic homogenization of the underlying services. For syntax analysis we developed a new, particular method, the so-called hierarchical regular expression parsing. Mediators work two-fold: They receive queries formulated in the global query language and create a query execution tree, thereby automatically generating additional query nodes that allow for the whole spectrum of query operations even on data sources with limited capabilities. From the bottom up, they translate the different local attribute models into the domain model. We developed several formal languages which are used for the specification of wrapper and mediator configuration files. Modifications, for example changes in the domain model or adjustments needed for reworked data sources, can be dynamically handled by editing the affected configuration files, without requiring a re-compilation of any system component. A new data source can easily be integrated by creating the appropriate specification files. All this gives us the extensibility and flexibility which is so much needed for a WWW-based meta system.

The integrator is the central system component, bridging the gap between user interface and internal components. It can quickly supply the user interface with results, dynamically growing the final result collection as it receives data from the mediators, thereby executing algorithms specially designed to work incrementally, e.g. the duplicate detection strategy. The integrator supports the user in the exploration of large result collections by providing numerous post-processing operations, like grouping or sorting. Since our architecture incorporates the Model-View-Controller pattern, the complete internal functionality is available for any user interface. A user can tailor the result to his needs by interactively operating on the result collection, for instance by exchanging the equivalence relation applied for duplicate detection. METALICA currently provides two views for displaying the final result collection, one a text-based HTML representation and the other a hierarchical tree view complemented with Hi-Cite functionality.

So far, we integrated several library catalogs, a publisher's catalog, and NCSTRL. Further services will be added. In order to also cover a broader spectrum of services, we currently evaluate the possible gain from including citation indices.

We also plan to add more views to the user interface, if possible by implementing adapters for existing visualization components (e.g. a 3D library metaphor). Furthermore, we currently prepare to conduct user studies on the different query and result interfaces to obtain insights into their relative performance and ease-of-use.

REFERENCES

1. S. Lawrence, C.L. Giles. Searching the World Wide Web, in Science, 280(5360), April 1998.

2. MetaCrawler, http://www.go2net.com/index_power.html

3. Highway61, http://www.highway61.com

4. E. Selberg and O. Etzioni. The MetaCrawler Architecture for Resource Aggregation on the Web, http://www.cs.washington.edu/research/metacrawler

5. Selberg, E. and O. Etzioni. Multi-service Search and Comparison Using the MetaCrawler, in: Proc. of the Fourth International World Wide Web Conference, Boston, MA, December 1995.

6. G.E. Krasner and S.T. Pope. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80, Journal of Object-Oriented Programming, 1(3):26-49, August 1988.

7. M. Wang Baldonado. An Interactive, Structure-Mediated Approach to Exploring Information in a Heterogeneous, Distributed Environment, Ph.D. Dissertation, Stanford University, December 1997.

8. Y. Arens, R. Hull, and R. King (eds.). Reference Architecture for the Intelligent Integration of Information, Program on Intelligent Integration of Information, ARPA, Draft Version 2.0, 22. August 1995, http://mole.dc.isx.com/I3/html/briefs/refarch.pdf

9. M. Baldonado, S. Katz, A. Paepke, C. Chang, H. Garcia-Molina, T. Winograd. An Extensible Constructor Tool for the Rapid, Interactive Design of Query Synthesizers, in: DL ´98, Pittsburgh, Pennsylvania, June 1998.

10. USMARC Format for Bibliographic Data: Including Guidelines for Content Designation. Cataloging Distribution Service, Library of Congress, Washongton, D.C., 1994.

11. S. Weibel, J. Kunze, C. Lagoze, M. Wolf. RFC 2413: Dublin Core Metadata for Resource Discovery, September 1998, ftp://ftp.internic.net/rfc/rfd2413.txt

12. K. Haller, H. Popst. Katalogisierung nach den RAK-WB: Eine Einführung in die Regeln für die alphabetische Katalogisierung in wissenschaftlichen Bibliotheken, Saur: München 1991.

13. Anglo-American Cataloguing Rules, Amendments 1993, Ottawa: Canadian Library Association; Chicago: American Library Association, ISBN 0838934315.

14. Y. Papakonstantinou, H. García-Molia, J. Widom. Object Exchange Across Heterogeneous Information Sources, in: Computer Society of the IEEE, Proceedings of the Eleventh International Conference on Data Engineering, Taipeh, Taiwan, March 1995, pp. 251–260.

15. E. Gamma. Design Patterns: elements of reusable object-oriented software, Addison Wesley, 1995, ISBN 0201633612.

16. Original Reusable Objects Inc., ORO-Matcher, Version 1.0.7., http://www.oroinc.com/

17. NCSTRL, http://cs-tr.cs.cornell.edu/

18. J.A. Hylton. Identifying and Merging Related Bibliographic Records, Masters Thesis, M.I.T. Department of EECS, 1996. http://ltt-www.lcs.mit.edu/lt-www/People/jeremy/thesis/

19. B. Shneiderman, D. Byrd, B. Croft. Sorting Out Searching. A User-Interface Framework for Text Searches, in: Communications of the ACM, 41(4), pp. 95-98, April 1998.

20. M. Wang Baldonado, T. Winograd, Hi-Cites: Dynamically Created Citations with Active Highlighting, in: CHI ´98, Los Angeles, CA, April 1998.

21. Karlsruher Virtueller Katalog, http://www.ubka.uni-karlsruhe.de/kvk.html

22. Dierolf, U. and Mönnich, M. Karlsruher Virtueller Katalog. Neue Dienstleistung im World Wide Web, in: Bibliotheksdienst, Heft 8/9, 1996.

23. A. Barth, M. Breu, A. Endres, and A. de Kemp. Digital Libraries in Computer Science: The MeDoc Approach, Lecture Notes in Computer Science 1392, Springer, 1998.

24. A. Paepcke, M. Baldonado, C.K. Chang, S. Cousins, H. Garcia-Molina. Building the Stanford Infobus: A Review of Technical Choices in the Stanford Digital Library Project, June 1998. http://www-diglib.stanford.edu/cgi-bin/WP/get/SIDL-WP-1998-0096

25. DealPilot, http://www.dealpilot.com/