A Scalable Component-based Architecture for Online Services of Library Catalogs

Marcus Flehmig

University of Kaiserslautern, Department of Computer Science[1]

Abstract. In recent years, more and more publications and material for studying and teaching, e. g. for Web-based teaching (WBT), appear "online" and digital libraries are built to manage such publications and online materials. Therefore, the most important concerns are related to the problem of durable, sustained storage and the management of content together with its metadata existing in heterogeneous styles and formats. In this paper, we present specific techniques and their use to support metadata-based catalog services. Such semi-structured metadata (represented as XML fragments), which belong to online learning resources, need efficient XML-based query support, scalable result set processing, and comprehensive facilities for personalization purposes. We discuss the associated problems, subsequently derive the concepts of a suitable architecture, and finally outline the realization by means of our prototype system that is based on the J2EE component model.

1 Introduction

More and more e-learning material is produced for online services. In general, it is integrated into portal systems [1] and can be used via the Web. However, searching online material is not easy. Therefore, digital libraries and online catalog services have been built. First of all, information about online material has to be collected and made queryable. For this purpose, a metadata repository with search (full text) or query (structured) facilities has to be made available. Ideally, such a repository should be based on standards like XML, RDF, or Dublin Core and should be interoperable with, e. g., OAI-supporting clients[2]. Furthermore, online catalog services should deploy state-of-the-art frameworks and architectural concepts to deal with scalability requirements. Particularly with regard to the very central task of search and query support scalability is essential. In addition, various kinds of statistics – related to the original queries, the most frequent subjects, the number of hits, or the variance of a distinct value distribution – are useful, e. g., to enable a comprehensive personalization concept, i. e., to provide navigational hints or help on choosing a good sorting or ranking method.

Finally, online catalog services stand for more than simple search engines. In fact, they provide search and query support but additional sophisticated information about the

1. *Author's address: University of Kaiserslautern, Department of Computer Science P.O. Box 3049, D-67653 Kaiserslautern, Germany. E-mail:* flehmig@informatik.uni-kl.de. *WWW:* http://wwwdbis.informatik.uni-kl.de/staff/Flehmig.

2. Open Archives Initiative: www.openarchives.org

query result is needed. They rely on standards and best-practices architectures to cope with scalability and interoperability needs. In more detail, the typical requirements, a suitable state-of-the-art component-based framework, specific technologies, the derivation of an appropriate architecture, and the outline of an applicable prototype system is presented in the next five sections.

2 Requirements

In order to provide an online catalog, it has to be filled with useful material. First of all, resources have to be found and identified. Some checks have to determine whether a resource contains suitable online material and is worth to be taken into account. If acceptable, it has to be indexed and reviewed. All of these steps should be supported by an appropriate management system, for which the underlying process is illustrated in Figure 1.

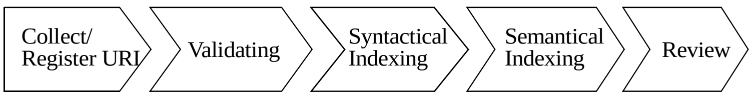

Figure 1 Basic Indexing Process

The indexing step is divided into two separated steps, because it is done by two different person (respectively roles), in general. Syntactical indexing provides simple metadata describing the online resource, for example, information about author, title, or format, whereas semantical indexing is related to the analysis of the content, for example, to build abstracts, identify subjects, keywords, or classification entries. To improve indexing quality, reviews have to be made by external experts who provide ideally XML-based surveys as a result.

By means of this process, the functional and non-functional requirements would be identified and presented in the next two paragraphs. The third paragraph of this section is about the need of XML-based processing.

2.1 Non-Functional Requirements

The overall indexing process has to be embedded into an appropriate infrastructure which provides administrative services for process control, backup, user administration, or rights management as well as for retrieval and evaluation tasks.

2.1.1 Infrastructural Tasks, Interoperability, and Support of Indexing Process

A couple of aspects have to be taken into consideration: different users and their roles have to be managed; authentication and authorization services have to be provided; metadata and so-called administrative attributes have to be stored under control of a persistence manager. Administrative attributes are associated with a distinct resource and represent its state to support the process of registration, validation and indexing. In addition, a user should be supported by automatic or semi-automatic tools. Furthermore, the ability to interoperate with other, for example, harvesting or OAI-based systems is essential.

2.1.2 Efficient Query Support

Regardless of all the above mentioned requirements, the principal task consists of efficient query support. Short response time, high throughput, low resource consumption, integration of full-text search and structured queries, as well as support of highly concurrent user queries are requirements summarized by the concept of efficient query support. Furthermore, the need of supporting so-called metaqueries. metaqueries are queries against result sets to provide the user with additional information concerning statistics about the most frequent subjects, number of hits, or the variance of a distinct value distribution. These statistics enable the system to offer navigational hints or help on choosing a meaningful sorting or ranking method. The underlying user model acts on the assumption that a user tries out some queries and refines a suitable query result by selecting a navigational hint to narrow the result set or by using one of the sort methods offered.

2.2 Functional Requirements

2.2.1 Data Model, Attribute Types and Domains

According to the applied requirements of the main project it has been necessary to define a special set of metadata attributes tailored to its special needs. For this reason, it has become possible to include special metadata attributes for online resources and peer reviews to deal with didactical issues and issues at law. In fact, for various reasons our project [7] has chosen Dublin Core and its meanwhile obsolete qualifier approach as a basis for defining a hierarchical scheme which has been guided by DCMES (Dublin Core Education Working Group). Nevertheless, other metadata schemes or systems like IMS/LOM, MathNet, Merlot, RENARDUS, ARIADNE, European School Net, or DLmeta have influenced our definition scheme. The natural-language-based definition as a whole can be found in [7]. An excerpt is shown in Table 1 to give an idea of the underlying data model, its different origins and data types. This data model is aware of elements and attributes and several data types. It offers the specification of single value elements, ordered lists, or unordered sets of elements. Attributes can be used to define additional data according to elements like language information.

Table 1: metadata Element Set and Structure: Example 1

Nr	Label	Origin	Multiplicity	Value Qualifier / Scheme
1.	Title	dc:	multi-valued, ordered list	structural
2.	Creator	dc:	multi-valued, ordered list	structural (avoiding mixed content)
2.a	Creator.Name	dcq:	single value	character string
2.b	Creator.Email	dcq:	multi-valued, ordered list	character string

Table 1: metadata Element Set and Structure: Example 1

Nr	Label	Origin	Multiplicity	Value Qualifier / Scheme
2.c	Creator. Organization	dcq:	multi-valued, ordered list	character string
16	Educational	akad:	single value	structural
16.a	Educational. Verified	akad:	single value	boolean
16.b	Educational. FieldOfStudy	akad:	multi-valued, ordered list	controlled vocabu-lary
17.	Audience	dc-ed:	multi-valued, unordered list	controlled vocabu-lary

2.2.2 Data Structure

All these metadata attributes are combined as a metadata record. Records, in turn, can be in relationship with other records. In this way, complex tree-like structures can be built. A learning resource record, for example, can be in relationship with a user comment or a peer review, in general, belongs to a distinct learning resource. Furthermore, it can be articulated that learning resources themselves are in relationship with other learning resources, e. g., if they occur in different formats (like PostScript or the widespread PDF format). The corresponding metadata attributes can be found in Table 2. Eventually these abstract data structures have to be transformed into a more formal description that can also be used to easily manage, store, exchange, efficiently query, or search them.

Table 2: metadata Element Set and Structure: Example 2

Nr	Label	Origin	Multiplicity	Value Qualifier / Scheme
13.	Relation	dc:	single value	structural
13.a	Relation. HasVersion	dcq:	multi-valued, unordered list	reference
13.b	Relation. HasPart	dcq:	multi-valued, unordered list	reference
13.c	Relation. HasFormat	dcq:	multi-valued, unordered list	reference
13.d	Relation. HasReview	akad:	multi-valued, unordered list	reference
13.e	Relation. HasUserComments	akad:	multi-valued, unordered list	reference

2.2.3 XML-based Representation

The deployment of XML and XML-Schema as a formal representation of the above described set of metadata allows a very sophisticated way of specifying underlying structures and using standardized methods and tools for processing and transforming data. It is feasible to describe the transformation of our specific metadata set representation (an example is given in Figure 2) into RDF or OAI-like structures in a very formal way to easily support interoperability with other systems, e. g., within library consortia. Furthermore, XML-Schema can cope with the needed range of element and attribute types. Therefore, an XML-Schema document could been derived from [7] which in turn establishes a formal basis for query support.

```
<learningresource guID="LR4162">
  <title>
    <main origin="intellectual" lang="DE">3-D Brain Anatomy</main>
  </title>
  <type>
    <document>Animation/Simulation</document>
    <interactivity origin="intellectual">false</interactivity>
  </type>
  <format>
    <mimetype>application/x-shockwave-flash</mimetype>
  </format>
  <identifier URI="http://www.pbs.org/wnet/brain/3d/index.html" />
  <language origin="intellectual">EN</language>
  <rights>
    <costs origin="intellectual">false</costs>
  </rights>
  <educational>
    <fieldOfStudy origin="intellectual">Psychologie</fieldOfStudy>
  </educational>
  <audience origin="intellectual">Grundstudium</audience>
</learningresource>
```

Figure 2 Example of XML Representation

2.3 Process Model: Element-based vs. Document-based Processing

The XML-based representation enables two distinct ways of processing the underlying data: element-based (fine-grained) or document-based (coarse-grained). As previously mentioned, the data model does not take the order of different elements into account, i. e. in terms of XML, the processing is data-centric. In contrast, multi-valued metadata attributes can be order-sensitive, which our system is aware of. However, the order is of less concern, in general. Therefore, it is conceivable that a single XML element could be the objective of authorization, processing, locking, updating, validation, or publishing. Such a fine-grained approach is very flexible but, in turn, complex and expensive. Hence, especially in enterprise applications a document-centric, i. e. coarse-grained, approach should be preferred.

As a consequence, a document, i. e. a textual XML-based representation of metadata, is built dynamically by composition of fragments. Each fragment describes a single learning resource and possibly has relationships to other resources. All these dependent resources are recursively added to the dynamically generated document. Such documents are fractions of the whole data base and are processed as a whole. They are conceptually very similar to the J2EE Transfer Object pattern [2]. The great advantages of this kind of processing are ease of locking and less communication effort. On the one hand, a client can receive in a single message all data requested and all the data needed to follow the relationships contained (like other prefetching approaches). On the other hand, the system has to cope with the problems of client-side updates and cache coherency. The use of an XML representation allows to specify a schema in a very formal way by using XML Schema[3]. The corresponding XML schema document represents the conceptual schema against which queries could be formulated. Herewith, the process model including defining, querying, and retrieving data is XML-based.

3 Basics

With the advent of object orientation (OO) many, old software engineering problems could be solved, but some problems still remain. OO and the corresponding languages only support fine-grained encapsulation or processing concepts. With the deployment of distributed processing, OO with its often small and very special classes and objects needs a reconsideration. Another problem could arise from the fact that objects have state, in general. Hence, scalability in large or data-intensive applications could become a problem. For this reason, components have emerged.

3.1 "Components are for Composition!"

Components could be defined in various ways, but a very appropriate definition is given by Szyperski: "A software component is a unit of composition with contextually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition." [12]. Accordingly, some general observations may be made. So, components are

- unit of deployment, reuse, and composition,
- interchangeable software parts, and
- have standardized interfaces,
- have explicit description of dependencies, and
- exposes enough of itself to permit some form of later-than-compile-time aggregation.

Components are a coherent package of software that can be independently developed and delivered as a unit, and that offers interfaces by which it can be connected with other components to compose a larger system. Its physical implementation is held in one place and components will often be larger-grained than the traditional object. They are more static, often have persistent storage and the need of an appropriate runtime environment (so-called, container).

3.2 Bifocal Perspective of State

Because the standard Web-communication protocol HTTP is a request-response protocol, individual requests of Web-based enterprise applications are treated independently. Consequently, such applications need a mechanism for identifying a particular client and the state of any conversation an application is having with that client. For this reason, state management truly becomes an issue at the application layer. Sometimes it is advised to push state management completely back into the database tier [4]. Therefore, methods

3. www.w3c.org/XML/Schema

have to be found to solve the problem of not propagating every change back into the database. Otherwise, the database layer has to be overcome with too many write operations and potentially locks, too.

A stateful component, by definition, is a component which maintains its state between calls to data members and methods. The term "state" refers to the current set of values of the various internal data members of the component. Thus, a stateless component, the other way around, is a component that doesn’t maintain its state between method calls. Nevertheless, an application can not exists without a state, but its component could be built in a stateless manner.

3.3 State Server

In the case of a stateless implementation, a so-called stateless component has to flush its state out after each component-level method finishes its execution. Accordingly, a component has to be associated with a client only during a single method call. After that call it could be released for potential use by other clients. With stateless implementations resource sharing is easy, because resources are exclusively used only by one component during a single concrete method execution. In the case of a stateful component, a resource like a database connection or a table lock is kept all over the time during a conversation between a client and server. The remaining question is where the so-called state has to be stored. If this service is realized at the component level, i. e. at the same level of abstraction, we might have scalability drawbacks. Components are dedicated to their corresponding runtime environment and have otherwise to migrate from one to another container, if the application is distributed over several nodes. Apparently, a more centralized storage solution seems to be more practical. As cited above, state could also be stored in a database. Conversational state alone could be stored outside a transactional context, but if a business method state or data is involved, we need a suitable transactional context to keep the database consistent. Therefore, these aspects have to be separated and handled independently in a transactional database system on the one hand and a state server for storing non-transactional conversational states on the other hand.

3.4 Patterns of Enterprise Application Architectures

The alteration from object orientation to component-based models accompanies with a shift in paradigm relating to the corresponding architectures. Traditional client/server architectures are displaced by enterprise application architectures. Even though enterprise application architectures could be considered as distributed n-tier client/server architectures, they are derived by the methodical appliance of engineering techniques and also by taking business application requirements into account. One of the most important techniques are design patterns. Patterns permit to describe best practices and good designs. In doing so, they formalize experience in a way that it is possible for others to reuse this experience. Furthermore, enterprise application architectures are closely related to Web services and service- oriented architectures (SOA) and characterized by the use of components, the deployment of efficient data access strategies, the development of clean and maintainable (Web) interfaces, and by the special consideration of performance and scalability.

4 Architecture Derivation

By now, we have specified some typical requirements of online library catalogs services and considered a few basic concepts. In the next paragraphs, we want to describe the derivation of a concrete architecture fulfilling the requirements described in Section 2 and using state-of-the-art best practice outlined in Section 3.

4.1 Separation of Concerns

The main focus of our approach lies on the separation of concerns. Therefore, distinct tasks have to be identified and have to be separated. First of all, we have separated the data (i. e. the learning object) from its metadata and the metadata from its related metadata (so-called meta-metadata or administrative metadata). As described in Section 2.2.2, the metadata is represented by XML and could be queryable by an appropriate descriptive language. To support reasonable queries, we have to support all the data types which are defined by the XML schema and a suitable set of operators (equal, less than, contains, etc.). The details of query executing and processing will be discussed in Section 4.2. Furthermore, beside the above separation of data and metadata we have made a strict distinction between three different main tasks: query processing, result set processing, and XML document processing. Each of these tasks is realized by a separate component which places us in a promising position to achieve better scalability.

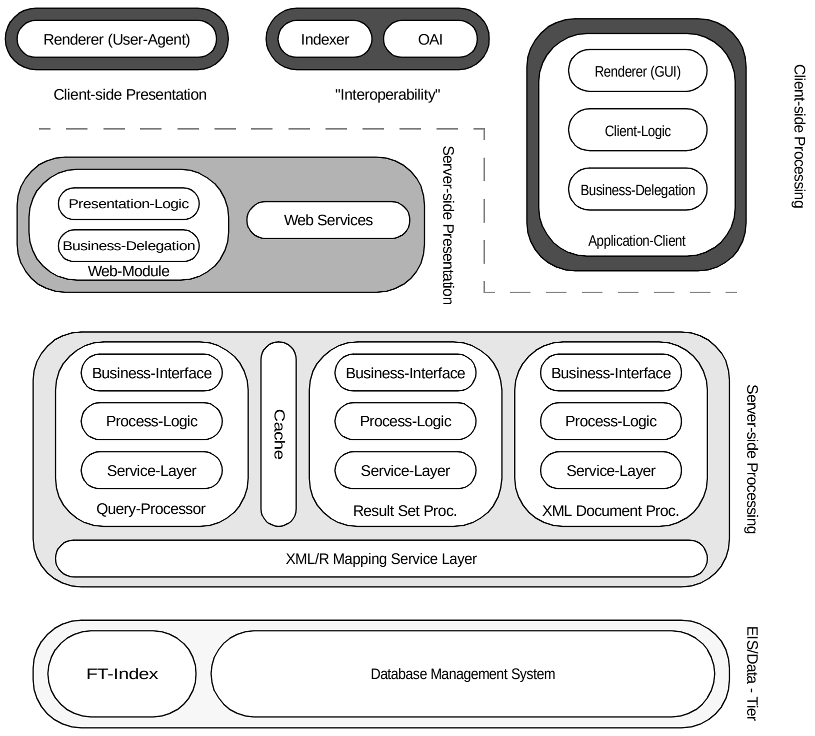

Figure 3 Architectural overview

As depicted in Figure 3, the separation of business and application (i. e., framework-specific) logic is essential for enterprise programming. In the bottom tier (enterprise information system tier) data is managed. Unstructured data (or learning objects) are managed and indexed by a freely-available full-text indexing system, whereas structured metadata is stored in a relational database management system. Therefore, we need a transformation between the XML model and the relational data model with simultaneous consideration of the integration of the full-text index. Support for this transformation can be found in the XML/R mapping-support layer which is located at the server-side processing tier. It builds the foundation of the higher server-sided components which include services for meta model access, domain value queries, or configuration management. The meta model provides information about the transformation from XML to relational and vice versa and will be presented in the next section in a more detailed manner.

The server-side processing tier contains more than the above mentioned mapping service. In addition, it provides components for query processing which is separated into three distinct steps. In the first step, a client query against the XML schema is transformed into an equivalent SQL query by the query processor. The result of such a query is cached and processed by a further component (result set processor). The XML documents which have been qualified for a query could be modified by the third component (XML processor) if needed. These components act like a controller in accordance with the MVC pattern [10]. Thus, they mediate between the underlying data tier containing the models and the presentational views.

Furthermore, they control the access to the three underlying (respectively contained) layers: the service layer, the process-logic layer and the business layer with its business interface. The business interface which is implemented by its corresponding controller represents the distinct component interface and helps to abstract from the particular application-specific or framework-specific implementation details. The implementation[4] of this interface realizes coarse-grained business methods by composition of less complex methods. The query engine, for example, provides methods to execute or refine queries and expects a descriptive query specification. The realization of the corresponding logic is done at the process-logic layer which uses and coordinates services provided by the underlying layer. The dynamic issues and the details of implementation are covered in Section 5.

The server-side presentation tier is located above the server-side processing tier. Web modules for supporting Web-based applications, for instance the Web-based search engine component, can also be found here as well as Web service connectors which provide interoperability to special clients (e. g. in the case of OAI support or deployment of external indexing or harvesting clients). Beside thin clients (i. e. user agents like Web browsers) fat clients are supported, too. These so-called application clients are used for several different tasks like harvesting, (semi-)automatical indexing, maintenance, editing or import/export. They can offer additional client logic which could hardly be provided by Web-based clients (for example, a complex graphical user interface). Application clients and Web modules directly access the server-sided components. Again, to abstract from particular application-specific or framework-specific implementation details, the business

4. Conceptually equivalent to SUN’s Session Facade pattern [10].

delegation pattern is deployed where all these details could be encapsulated. The business delegation pattern implementation builds a single point of communication between Web modules located in the server-side presentation tier and the various controllers in the server-side processing tier.

4.2 Query Engine and Processing

As previously mentioned, the library service application could be characterized as data-intensive and XML-based. Due to the combined use of XML and RDBMS, a mapping from the external XML representation to the internal relational DB-schema is needed. Such a mapping must not be static; otherwise, schema evolution becomes impossible. For this reason, we decided to pursue a meta model-based approach. Each query against the conceptual XML schema has to be transformed into a corresponding SQL query against the relational schema. Furthermore, several requirements have to be fulfilled by such a meta model. The mapping defines for each XML path, i. e. for each path to several content nodes, in which SQL attribute and in which table its content is stored. Additionally, the XML data type, the SQL column type, whether it is an attribute or element, information about its domain, and a flag whether an order id has to be managed are stored in an additional relational structure. The order id represents the position of an element in its document representation and is managed together with the other attributes in the corresponding tables. This structure and its data alone isn't sufficient for answering queries.

XML documents are spread over several tables and those tables have to be joined together to support combined predicates and to determine to which document (learning object) a qualifying tuple belongs. Therefore, all referential constraints are made explicit by specifying them in the meta model. To avoid recursive expressions during calculation, a dependency between two elements, the path from one element to another, is stored in a separate structure, too. Therefore, only a simple query is needed to decide which tables have to be joined. Henceforth, we are able to direct queries against an XML schema by transforming them into queries against a relational schema.

4.2.1 Query Model

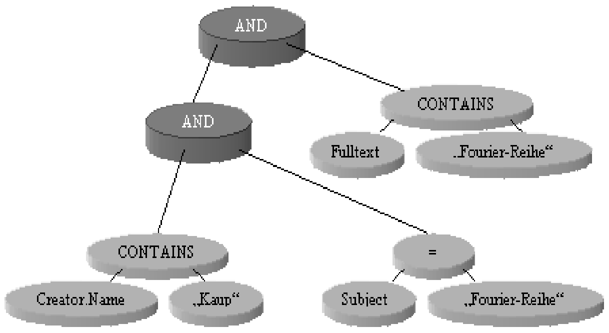

Figure 4 Tree Representation of a Typical Query

Figure 4 shows a tree-based representation of a typical query generated by the Web-search component. At the programming level, a query is represented by a corresponding object structure. It is possible to combine simple predicates to build more complex expressions. Thereby, a significant subset of XPath[5] is supported. The so-called que-

5. www.w3c.org/TR/xpath

ry tree is sent to the query processor and, with the help of the meta model, it is transformed into an SQL query.

4.2.2 Query Execution and Transformation

As shown in Figure 3, the query processor has its own front component. Clients can only interact with the controller in which the other components are embedded. It coordinates the query processing by decomposing a request into several subtasks, arranging them along a pipeline, as well as executing and managing the pipeline. At first, the query tree is parsed into the internally used logic tree and simultaneously validated syntactically. The logic tree supports several transformations. Therefore, in a second step the logic tree is simplified. Based on this optimized logic tree, a query plan is derived and optimized.

The full-text index is integrated at this early point of time, because full-text processing is very expensive. In the case of synchronous processing, it takes 80 percent of the overall query execution time to search the index and fully materialize the result, as dedicated measurements have shown. But, it can done very well in parallel during query plan generation and optimization. Thus, the earlier the full-text search is started, the greater is the benefit, because both tasks are completely independent. By now, both parallel tasks are synchronized immediately before SQL query execution. Therefore, the cached results can be easily mediated with the remaining part of the query. The optimized query plan is used to generate an SQL statement to evaluate the query sent by a client. The result of this query is also cached. Each further operation works on this cache copy identified by a so-called result set handle. However, it is feasible that query execution as well as full-text search is deferred until results are really needed by a client, which is planed to be implemented in a future version.

4.2.3 Result Set Processing

As a consequence of the above mentioned separation of concerns, the query processing is separated from the result set processing. This is achieved by caching the result set and providing a result set handle for identification purpose. But caching is also necessary to accomplish some reordering of the result set, because the full-text search engine does not support comprehensive sorting capabilities. However, caching even offers some more advantages. To determine the number of hits, the most frequently used terms or subjects, or to make helpful hints to determine appropriate search criteria by building the variance of distinct columns, it is not necessary to rerun the whole expensive query. Beyond these metaqueries it is feasible to use the cached results for query refinement, for reuse during processing of subsequent queries and to provide a so-called stateless cursor.

4.2.4 Stateless Cursor Concept

In general, the communication between a client and a database server is stateful, i. e., there exists a conversational state between different methods calls. In data-intensive Web applications where a significant number of objects or database tuples has to be managed, it is essential to solve the problem of sampling the set, i. e. building blocks of data, and mapping these samples to the hypertext document structure of the Web. SUN's fast-lane reader pattern [2] is only one solution of this problem. However, it is not feasible to hold a database cursor open all the time a user navigates through the result set pages, because open database connections are too expensive. Furthermore, in the case of stateful components, there exists a component for each single client request. As mentioned in Section 3.2 and Section 3.3, an implementation could also be realized stateless and state can be managed by a distinct component or the database system. In our approach, the result set of a client query is cached, i. e., it is stored into a separated (temporary) table of a relational DBS together with additional administrative data. Hence, concurrent queries and locks are no problem any longer and every subsequent query transforming or refining the result set could be executed locally in the cache. This idea of a stateless cursor and additional realization details are described in Section 5.

4.3 Scalability

The trichotomy of the overall query process comes along with a very positive impact. By deploying such an architecture, we can cope with the typical workloads of digital library usage pattern and achieve better scalability. Typical workloads rely on the assumption that a user tries out some queries and refines a suitable query result by selecting a navigational hint to narrow the result set or by using one of the sort methods offered. Additionally, information about the current query result should be presented to the user, e. g. the total amount of hits, how many documents are classified by a distinct subject, or information about the most frequent keywords. Though, evaluating such information is expensive. In most cases, a query has to be repeated more than one time to build suitable aggregated information about distinct columns of a result set. In respect of our requirements, a query has to be executed at least four times to aggregate all of the needed information. As previously mentioned, intermediate results are cached and reused for optimization purpose. Incorporating the cache to evaluate metaqueries allows to specify very simple and inexpensive queries which get along with a few simple joins and which disburden the underlying DBS. Furthermore, our approach yields another benefit regarding sort support. Consecutive queries differing solely in the applied sort criteria could be answered without the necessity to repeat the query. As a consequence, only the order operation itself has to be executed and execution time could be halved, on average. Whensoever the user is using a former sort criteria, the previously calculated and cached results could be reused without any additional effort, regardless of ascending or descending order.

In addition to caching operational data of a distinct user query, conversational state is stored, too. For this reason, a stateless component architecture becomes feasible. The state is managed outside of the component and could be restored before processing is continued regardless where the component resides, i. e. in the case of distributed processing, each node could answer any request. Furthermore, the DBS is unburdened from unnecessary queries. Initial queries are processed by the query engine. As well, the query engine itself is implemented in a stateless manner and can be distributed, too. Result set processing is done by a separate component which operates on top of the cache. For this reason, subsequent queries are performed locally and, thus, do not stress the underlying database system.

5 Realization

The Java 2 Platform – Enterprise Edition (J2EE) builds the standard in the domain of Java-based multi-tier enterprise applications. Therefore, the Java 2 Platform – Standard Edition (J2SE) is extended by many additional services and technologies for transaction management, security, persistence management, resource pooling, life cycle management, remote client connectivity, naming service, or deployment. Furthermore, Enterprise JavaBeans (EJBs), Java Servlets, Java Server-Pages (JSP) are introduced as server-sided components [11].

5.1 J2EE Application Model and its Components

J2EE is influenced by the common 3-tier architecture and the separation of presentational aspects, business or application-logic and data access or management tasks. In more detail, it distinguishes between client-side presentation, server-side presentation, server-side business-logic and enterprise information systems (EIS), as depicted in Figure 5.

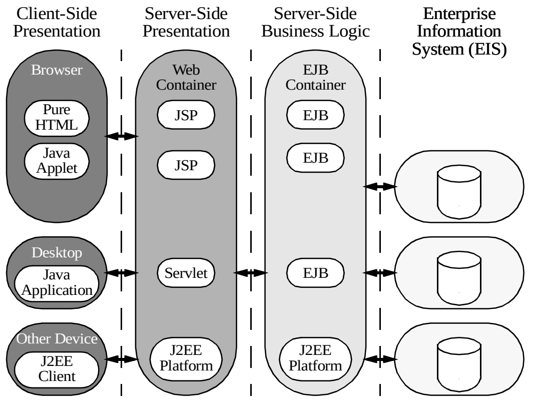

Figure 5 J2EE Architecture Overview (SUN)

All J2EE components depend on the runtime support of a system-level entity called a container. Containers provide components with services such as life-cycle management, security, deployment, and threading. Because containers manage these services, many behavioral properties of components can be declaratively customized when the component is deployed in the container [11].

J2EE supports two types of server-side enterprise components. First, the Web components, i. e. servlets and JSPs, reside in the Web container. Second, EJBs belong to the EJB Container and the so-called server-side business logic. EJBs as reusable software artifacts are part of a flexible and powerful component model in the domain of distributed enterprise application systems. There exist three classes of EJBs: message, entity, and session beans. Entity EJBs represent persistent business objects, session beans are used for the coordination and control of such business objects, whereas messages EJBs could be used for realization of asynchronous behaviour. Hence in most instances, session beans realize the business logic.

5.2 Stateless Cursor Implementation

The query execution is separated into three distinct tasks. The responsible components are realized by using session EJBs. Each controller of the query processor, result set processor and XML processor is realized by a stateful session bean, stateless session bean or both. In the latter case, the deployment descriptor settings determine which alternative is used. This flexibility could be very useful to support different optimization strategies, e. g., there are two implementations of the result set processor – one for the support of an editing tool and one for the support of Web-based queries. The first is realized in a stateful way and offers, for example, a broad range of sort criteria, whereas the second one tries to minimize the number of open connections and maximizes the number of possible parallel requests by using a stateless implementation and the stateless cursor concept. In the case of statelessness, resource sharing and pooling is much easier than in the stateful case.

5.2.1 Result Caching

The meta model-based approach postulates the transformation of each XML-based query into a corresponding SQL query against a relational schema. Therefore, a query tree is sent to the query processor and with the help of the meta model it is transformed into an SQL query and its result is materialized in the cache. Full-text queries are integrated by calling the full text engine and materializing the results, too. Because both result sets are represented using the same data model, they could be easily mediated. Each consecutive operation works on this mediated and cached copies identified by a result set handle. Therefore, it is not necessary to repeat the whole query to process metaqueries. Results are cached by an RDBMS and stored into (temporary) tables.

5.2.2 Distinct Sort Support

The result set tuples of a temporary cache table are ordered and augmented by a sequence number. There exist four predetermined, but adaptable sort criteria and for each of them a sequence is stored. The overhead could be neglected compared to the benefit. An order operation has to be processed only once. If a user tries out several sort criteria (e. g. ranking, quality, or date of creation) to find a suitable order, the sort order could be easily selected. Ultimately, these sequences could be simply used for range queries that are required by the Web components for representing the results in smaller groups of objects fitting into a single page. In fact, navigation is very simple and efficient. Other sort criteria are available via the stateful alternative which can be seamlessly switched to.

5.2.3 Benefits

The main benefit of the stateless cursor concept with its caching lies in better scalability. On the one hand, scalability is improved by the J2EE multi-tier architecture. On the other hand, middle-tier components could be very easily distributed over various server nodes. The conversational state is managed by the cache component, i. e., it is stored in a centralized database system. The materialization of query results could also be used for query refinement. By now, the cache tables are managed by the backend DBS, i. e. at the EIS tier, because it is necessary to combine (join) the cache tables together with other operational data to refine the original query and to process metaqueries. But it is also feasible to manage state and cached results at the middle tier by an in-memory database system and to replicate additional data originated by the backend database system. This loose coupling of components allows to easily exchange or expand the components.

6 Related Work

The World Wide Web (WWW) provides a lot of different search engine services[6] or appropriate products[7] for nearly all domains including digital libraries. But, an online service for providing library catalogs has to fulfil more requirements than mere information retrieval tasks. At least, it must provide support of managing and querying structured metadata similar to directory services[8]. Furthermore, support of underlying indexing

6. www.google.com, www.altavista.com

7. www.aspseek.org

processes (manual or automatic) is essential. The different indexing process approaches mostly differ in the underlying metadata structures. Therefore, our approach of providing highly scalable and personalizable digital library services certainly has to consider standards like IMS or Dublin Core. But none of the existing standards alone can cope with all the facets of our requirements concerning online teaching or learning resources. For this reason, we have to define an appropriate set of metadata attributes by ourselves to facilitate their use. The underlying generic system architecture of our approach, however, is more or less independent of such a set of metadata attributes.

MILESS [8] is built on top of an existing document management system and could be used to manage resources and the corresponding metadata whose structure is based on the Dublin Core standard. Its successor called MyCoRe can be deployed for the development of Digital Library and archive solutions. Adjustability, extensibility, and open interfaces are fundamental design premises of MyCoRe, but at the moment it is dedicated to only a single database management system. Its import and export format for the describing data will be XML, too [9]. MyCoRe supports rich queries facilities, whereas the DBLP project[9] is focused on scalability. The DBLP server provides bibliographic information of major computer science journals and proceedings, indexes currently more than 470000 articles, and contains several thousand links to home pages of computer scientists (November 2003). DBLP uses an XML-based metadata structure and is based on a file system. However, extended search facilities are provided by an additional tool [13].

In contrast, our approach is able to cope with a more comprehensive set of metadata attributes and uses standardized techniques as well as a component-based framework (J2EE). It is very flexible concerning the underlying metadata schema. In addition, automatic indexing services are integrated in our system. Thus, interoperability, scalability, process-oriented metadata management, and the integration of full-text search and structured queries are the outstanding characteristics of our approach.

7 Conclusions and Outlook

In this paper, a scalable component-based architecture for online catalog services has been presented that can be used in the domain of digital libraries. For this purpose, we have discussed the requirements of the XML represented metadata management and its realizing components. The innovative aspect of our approach could be found in the segmentation of the query process. Hence, sophisticated caching becomes feasible. It is not necessary any longer to rerun expensive queries which could additionally include an unstructured search to answer the so-called metaqueries and to accomplish the mentioned requirements of personalization. The presented concepts have been validated by realizing a prototype system[10], that is based on the briefly outlined J2EE component model. Nevertheless, some implementation details of our prototype could be improved. By now, the data access layer and the service layer are not as strictly separated as they should. The next step aims at the deployment of asynchronous messaging based on J2EE Message EJBs

8. www.yahoo.com, www.dmoz.org
9. dblp.uni-trier.de
10. www.akleon.de (at the moment only available in German)

[5]. Further improvements can be expected by better cache integration. Thus, it is feasible to realize query refinement by utilizing already cached data. Furthermore, in-memory management of the cache at the middle-tier should be considered which is similar to Javlin (Objectstore[11]) or [14].

References

[1] Flehmig, M.: Integration und Personalisierung − Zur Realisierung essentieller Aspekte in Portal-Systemen, in: Proceedings of the International GI/OCG-Informatik 2001 Conference, Wien, Sept. 2001, pp. 895-901 (in German).

[2] Alur, D., Crupi, J., Malks, D.: Core J2EE Patterns − Best Practices and Design Strategies, Prentice Hall / Sun Microsystems Press, 2001.

[3] Bollacker, K., Lawrence, S., Giles, C.L.: A System for Automatic Personalized Tracking of Scientific Literature on the Web, in: Digital Libraries 99 − The Fourth ACM Conference on Digital Libraries, ACM Press, New York, 1999, pp. 105-113.

[4] Intel e-Business Center: N-tier Architecture Improves Scalability, Availability and Ease of Integration, White Paper, 2001.

[5] Kounev, S., Buchmann, A.P.: Improving Data Access of J2EE Applications by Exploiting Asynchronous Messaging and Caching Services, in: Proceedings of the 28th VLDB Conference, Hong-Kong, 2002, pp. 574-585.

[6] Lawrence, S., Bollacker, K., Giles, C.L.: Autonomous Citation Matching, Proceedings of the Third International Conference on Autonomous Agents, Oren Etzioni, ACM Press, New York, 1999.

[7] META-AKAD: Metadata Element Set and Structure, in: Technical Report, University of Kaiserslautern, University of Regensburg, September 2003 (63 pages).

[8] Miless: The Miless Homepage, http://miless.uni-essen.de/, University of Essen, Germany, 2003.

[9] MyCoRe: The MyCoRe Project Homepage, http://www.mycore.de/engl/index.html, 2003.

[10] Sun Microsystems: Java BluePrints − Model-View-Controller, http://java.sun.com/blueprints/patterns/MVC-detailed.html, 2002.

[11] Sun Microsystems: Java 2 Platform, Enterprise Edition − Overview, http://java.sun.com/j2ee/overview.html, 2003.

[12] Szyperski, C.: Component Software − Beyond Object-Oriented Programming, ACM Press, New York, 1998.

[13] Witten, I.H., Moffat, A., Bell, T.C.: Managing Gigabytes: Compressing and Indexing Documents and Images, Morgan Kaufmann Publishing, San Francisco, 1999.

[14] Altinel, M., Bornhövd, C., Krishnamurthy, S.,Mohan, C., Pirahesh, H., Reinwald, B.: Cache Tables: Paving the Way for an Adaptive Database Cache, in: Proceedings of the 29th VLDB Conference, Berlin, 2003, 718-729.

11. www.objectstore.net