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

Marcus Flehmig

University of Kaiserslautern, Department of Computer Science, P.O. Box 3049,
67653 Kaiserslautern, Germany
flehmig@informatik.uni-kl.de

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, their use to support metadata-based catalog services deploying XML, and finally derive the concepts of a suitable component-based architecture.

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. 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. 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 query result is needed, which is presented in more detail in the next sections.

2 Requirements

To provide an online catalog, it first has to be filled with useful material. In an initial step, resources have to be found and identified. If acceptable, they have 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 Fig. 1. 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 term “efficient query support". In addition, various kinds of statistics – related to the original queries, the

N. Koch, P. Fraternali, and M. Wirsing (Eds.): ICWE 2004, LNCS 3140, pp. 396–401, 2004.

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.

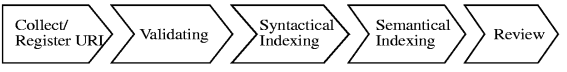

Fig. 1. Basic Indexing Process

The list of requirements of our project made it necessary to define a special set of metadata attributes tailored to its special needs. For this reason, we included 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 [2] has chosen Dublin Core and its meanwhile obsolete qualifier approach as a basis for defining a hierarchical scheme. This data model is aware of elements and attributes and a number of data types. It offers the specification of single-value elements, ordered lists, or unordered sets of elements. Metadata records are in relationship with other records. In this way, complex tree-like structures may result. A learning resource record, for example, may be in relationship with a user comment, a peer review, or again a learning resource, e.g., if the material occurs in different formats (like PostScript or the widespread PDF format). Deploying XML as formal representation makes it possible to describe the transformation of our metadata set representation into RDF or OAI-like (Open Archive Initiative) structures in a very formal way to easily support interoperability with other systems, e.g., within library consortia. The XML-based representation enables two distinct ways of processing the underlying data: element-based (fine-grained) or document-based (coarse-grained). Hence, especially in enterprise applications a document-centric, i.e., coarse-grained approach should be preferred. Therefore, the process model including definition, query, and retrieval of 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 concepts. Hence, scalability in large or data-intensive applications could become a problem. For this reason, components have emerged. 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 comparing to the traditional concept of objects. Because the standard Web-communication protocol HTTP is a request-response protocol, individual requests of Web-based applications are treated independently. 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 [3]. With stateless implementations, resource sharing is easy, because they are exclusively used only by one component during a single method execution. Apparently, a centralized storage for component state seems to be very practical. The alteration from object orientation to component-based models is accompanied by a shift in paradigm related 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.

4 Architecture Derivation

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 administrative metadata). 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. As depicted in Fig. 2, 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 system (RDBS). 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 located at the server-side processing tier. It builds the foundation of the higher-level server-sided components which include services for meta model access, domain-value queries, or configuration management.

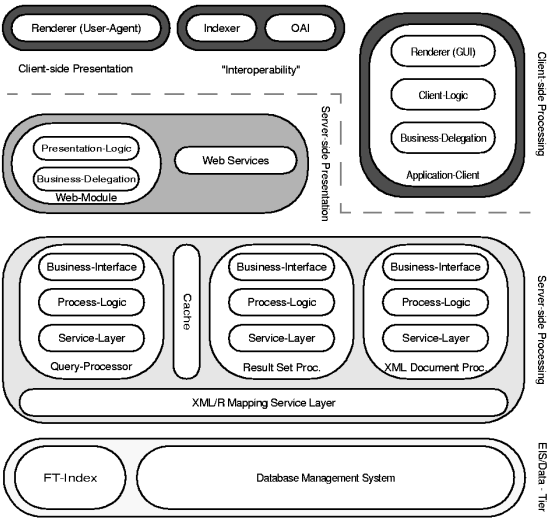

Fig. 2. Architectural overview

In addition, the server-side processing tier provides components for query processing which is separated into three distinct steps. In the first step, a client query against an XML schema is transformed into an equivalent SQL query by the query processor which provides methods to execute or refine queries and expects a descriptive query specification. The result of such a query is cached and processed by an additional component (result set processor). The XML documents 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 [4]. Thus, they mediate between the underlying data tier containing the models and the presentational views. The business interface implemented by its corresponding controller represents the distinct component interface and helps to abstract from the particular application-specific or frameworkspecific implementation details. The implementation of this interface realizes coarsegrained business methods by composition of less complex methods and is conceptually equivalent to SUN's Session Facade pattern [4].

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, deployment of external indexing, or harvesting clients). Beside thin clients (i.e., user agents like Web browsers) fat clients for maintenance and indexing tasks are supported, too. Again, to abstract from particular application-specific or framework-specific implementation details, the business 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.

As a consequence of the above mentioned separation of concerns, query processing is separated from 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 entire, typically very expensive, query. Beyond these so-called metaqueries it is feasible to use the cached results for query refinement, for reuse during processing of subsequent queries and for provision of a so-called stateless cursor. 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. In the case of stateful components, there exists a component instance for each single client request. An implementation has to be realized in a stateless manner and state can be managed by a distinct component or a RDBS. In our approach, the result set of a client query is stored into a separated (temporary) table of a RDBS together with additional administrative data (e.g., cursor position). Hence, concurrent queries and locks are no problem any longer and every subsequent query transformation or refinement of the result set could be locally executed in the cache.

5 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 patterns 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. Regarding our requirements, a query has to be executed at least four times to aggregate all of the information needed. Incorporating the cache to evaluate, the above mentioned, metaqueries allows to specify very simple and inexpensive queries which get along with a few simple joins and which relieve the underlying RDBS. Furthermore, our approach yields another benefit regarding sort support. Consecutive queries differing solely in the sort criteria applied could be answered without the necessity to repeat the query.

In addition to caching operational data of a distinct user query, conversational state information 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. 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, initial queries are processed by the query engine and subsequent queries are performed locally. As a consequence, subsequent queries do not stress the underlying database system.

6 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. In contrast to search engine services (e.g., Google or Altavista), appropriate products (e.g., ASPseek), directory services (e.g., Yahoo or DMOZ), or other digital library systems (e.g., MyCoRe), 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. 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 [5]. Therefore, the innovative aspect of our approach could be found in the segmentation of the query process. As a consequence, sophisticated caching becomes feasible. It is not necessary any longer to rerun expensive queries which could additionally include unstructured search to answer metaqueries and to accomplish the requirements of personalization. The concepts presented have been validated by realizing a prototype system[1]. Furthermore, in-memory cache management at the middle-tier may be considered which is similar to the approach described in [6].

[1] www.akleon.de (at the moment only available in German)

References

1. Flehmig, M.: Integration and Personalization – On Realization of Essential Aspects of Portal Systems, in: Proceedings of the International GI/OCG Computer Science Conference, Vienna, Sept. 2001, pp. 895–901 (in German).
2. META-AKAD: Metadata Element Set and Structure, in: Technical Report, University of Kaiserslautern, University of Regensburg, September 2003 (63 pages).
3. Intel e-Business Center: N-tier Architecture Improves Scalability, Availability and Ease of Integration, White Paper, 2001.
4. Alur, D., Crupi, J., Malks, D.: Core J2EE Patterns – Best Practices and Design Strategies, Prentice Hall / Sun Microsystems Press, 2001.
5. MyCoRe: The MyCoRe Project Homepage, `http://www.mycore.de/engl/index.html`, 2003.
6. 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, pp. 718-729.