Enhancing the Functionality of the Web

P. Kutschera, R. Rantzau
University of Stuttgart
Institute of Parallel and Distributed High-Performance Systems (IPVR)
Breitwiesenstr. 20-22
D-70565 Stuttgart
Germany
E-mail: {kutschera,rantzau}@hermes.informatik.uni-stuttgart.de

Abstract

This paper presents an approach to combine the World Wide Web and database systems in a way that all of the Web based resources are stored and managed by the database system. Access to the information located in the database system is based on an URL notation which will be described in detail. Addionally, the database system is used to store all of the necessary information that is used to overcome the most serious limitations of the World Wide Web like existence of the "lost in hyperspace" syndrome, invalid links as well as the limited search facilities.

1 Introduction

The World Wide Web (WWW) is probably the most popular information system today. Its popularity is mainly based on the appealing graphical user interface of the browser, the availability of the software on almost every hard- and software platform as well as the huge framework of resources that are accessible world wide. But, despite of its popularity, the WWW comprises some well-known limitations like the "lost in hyperspace" syndrome, the existence of invalid links and the limited search facilities. Another feature of the World Wide Web which is currently missing is the ability to access information located in a database system in a standard fashion. These shortcomings limit the use of the World Wide Web in many different applications like on-line information services or learning environments where most of the relevant information is stored in a database and the integrity of the online information is very important. However, in combining the features of the World Wide Web with the functionality of database systems, it is possible to overcome these limitations and "enhance the functionality of the Web".

The approach presented in this paper combines the World Wide Web and database systems in a way that all of the Web based resources (documents, images, audio or video clips, Java applets, etc.) as well as any additional meta data are stored and managed by the database system. The meta data comprise all of the information that is necessary to provide link integrity, content-based searches and navigational aid for online access. Access to information located in the database system is based on an appropriate URL notation which will be introduced in this paper and can be used independent of the underlying database environment. Basically, this paper extends and further refines the work that is presented in [7].

2 Related Work

Today, almost every database vendor sells its own proprietary product that is capable of accessing its own database system via WWW. Most of these solutions use the API of the Web servers to provide this additional functionality and rely on vendor-specific extensions to the HTML language to include database queries directly into HTML pages. Based on these so-called application pages which are either stored in the Web server's file system [9] or in the database system, it is possible to enhance HTML pages with database queries. While almost all of these products are only capable of including relational or object-oriented queries into HTML pages, only one of them [3] is capable of accessing audio or video content, images, HTML documents or Java applets residing in the database as well. Another possibility for accessing a database system is to use a CORBA based Web environment with an integrated ORB like as it is described in [1]. In summary, it may be said that the database vendors provide a proprietary access to their own database product, but do not try to solve the problem of invalid links or the limited search capabilities for Web based resources.

The idea of storing information about hyperlinks in a database system to provide link integrity has been used in many hypermedia systems (see [5] for example). In [11] this approach has been applied to the World Wide Web, but because all of the Web based resources are still located in the Web server's file system, only a limited form of link consistency can be guaranteed in comparison with the approach presented in this paper. The same kind of limitations like those of the World Wide Web have already been encountered in other hypermedia systems as well and led to the development of the so-called second generation hypermedia systems like Hyper-G [8]. Because of the different hypertext models and communication protocols of these hypermedia systems, these solutions cannot be applied to the World Wide Web.

3 Concepts

3.1 Link integrity

In the World Wide Web, a link denotes a forward reference from a HTML document (link source) to any type of multi- or hypermedia content (link target) and is embedded directly into its source document. Thus, the term link is here not only used to denote hyperlinks, but for all kinds of references within a HTML page where an URL is permitted. The problem of the dangling links results from the WWW's hypertext model that only uses forward references for links without maintaining the affiliated backward references. Thus, if a link becomes invalid due to changes of the target resource, there is no possibility to adapt the source document of the link according to these changes. The only possibility to detect an invalid link is to scan the HTML pages in the local file system using tools like the Web crawler on a regular basis and analyze the existing link structure. By combining the World Wide Web and database systems in a way that all of the Web based resources including documents, images, audio or video clips as well as Java applets are stored and managed by the database system, link integrity can be guaranteed by extracting the relevant link information during insertion of a HTML document and storing them along with the missing backward references in the database. Afterwards, this information has to be kept up to date by monitoring all of the changes that are applied to the document structure. Prior to retrieving a HTML document from the database, the validity of the links within the document has to be checked and invalid links have to be removed to shield the user from accessing dangling links.

3.2 Additional search facilities

Although there are many different resources available on the World Wide Web, it is very difficult to locate resources about a certain topic. Currently, either a directory service such as Yahoo or one of the existing search engines such as AltaVista, Web Crawler, etc. [12] can be used. While this approach is feasible for searching larger parts of the Web, these methods are rather ineffective when someone wants to find relevant material about a certain topic at a particular Web site. Thus, by using a database system it is possible to extract and store additional meta data like keywords, headings, table or figure captions of HTML pages in the database prior to storing the HTML pages itself. Now, the built-in query language of the database system cannot only be used for full-text searches, but also to search specific parts of HTML documents very efficiently.

3.3 Navigational aid

The World Wide Web provides the user with a navigational interface along the existing hyperlinks. While in principle, it is a very elegant way to link related information together, it has been observed that users often feel disoriented after following a few hyperlinks. This phenomenon which is often called "lost in hyperspace" results from a missing navigational aid similar to a table of contents in a book or a guided tour through the Web site. Based on the information about the existing link structure, it is possible to display the entire hypertext structure by a frontend tool in form of a map to indicate the users' current location while navigating through the hyperspace.

3.4 Accessing database content

As it has been demonstrated so far, by combining the WWW and database systems it is possible to overcome the most serious limitations of the World Wide Web. Because a lot of different Web applications benefit from eliminating these shortcomings, they should be accomplished in a way that is independent of the underlying database and Web environment. Unfortunately, references to resources located in a database system cannot be embedded into HTML pages as easily as references to resources located in the Web server's file system due to the fact that they cannot be denoted by an URL. Thus, an appropriate URL notation is developed to denote resources located in the database as well. The advantage of using an URL, besides the fact that it is independent of the underlying database environment, is that all of the information that is accessible via World Wide Web is treated in a uniform fashion. Thus, the administration and development of the appropriate HTML pages is much simplified.

4 Combining databases and the World Wide Web

This section gives an overview in which way a World Wide Web based access to database systems can be accomplished and introduces the URL notation that has been developed for denoting information located in a database system. Finally, all of the details are revealed how link integrity, additional search facilities and navigational aid can be accomplished by using additional meta data.

4.1 Architecture

When developing a World Wide Web based access to database systems, various approaches have to be considered that all differ in regard of the efficiency that can be achieved and the necessary software environment which is required. Generally, it is possible to use CGI scripts, extend the functionality of the Web server or using a CORBA based approach [1] to access the database system. Although, CGI scripts provide access to external data sources in a standard fashion, they lack efficiency due to the overhead of starting a new process, namely the CGI script, for every single database request. An ORB based Web environment, on the other hand, offers a lot of interesting possibilities for distributed applications, but is currently not the state of the art. Thus, the only possibility that is left is to extend the functionality of the Web server itself. Although, currently every Web server offers its proprietary API for extensions, this approach lacks portability between different platforms as well as products. Fortunately, a new generation of extensible Web servers [2, 4] already exist that can be extended using the Java programming languages. These Java based extensions of the Web server which are called servlets are in fact server-side equivalents of Java applets. Compared to the other solutions they offer an enhanced portability, efficient database access as they are loaded during startup of the Web server and the security model of Java applets. Additionally, servlets may access the database by using the JDBC (Java Database Connection) [6] interface which is independent of the underlying database environment. In summary, for portability and security reasons the servlet approach has been chosen in favor of the other approaches described above.

After the general framework has been introduced, the interaction between the different components of the architecture as it is depicted in figure 1 will be described. The

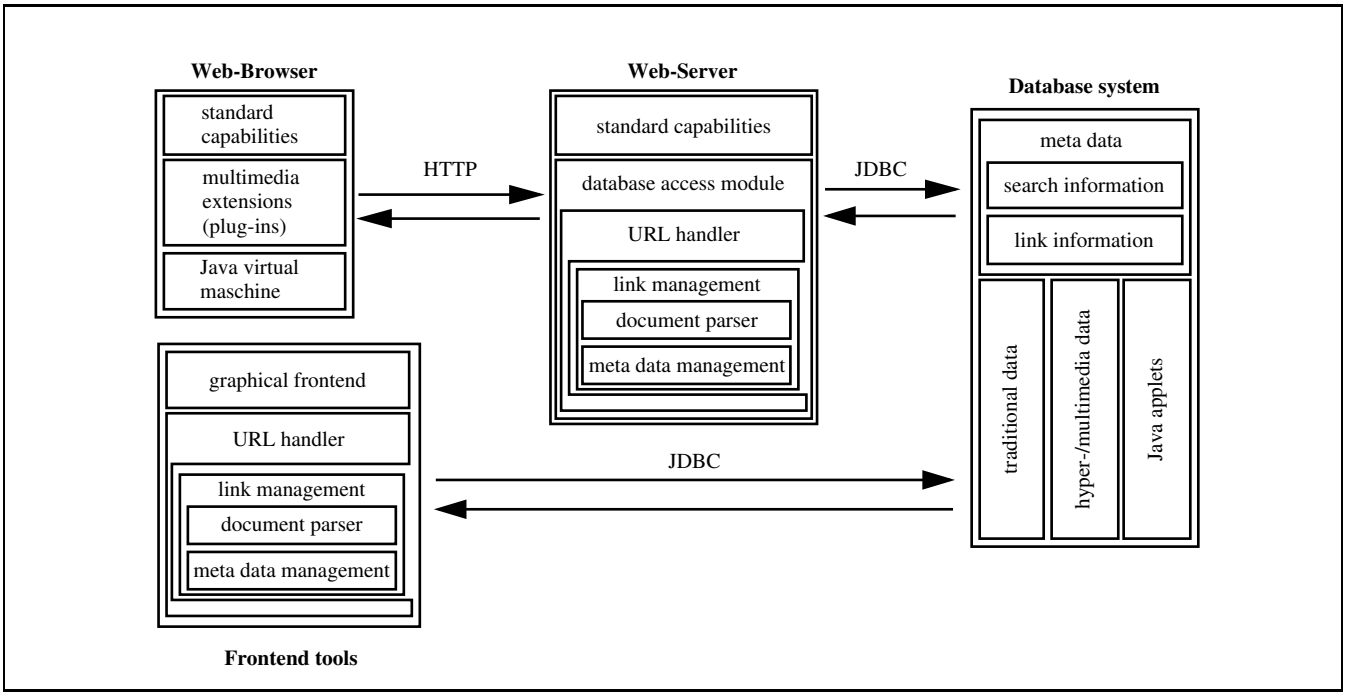

Figure 1: The general component architecture

retrieval of information located in the database system is possible with any ordinary Web browser whereas insertions, deletions, modifications and migrations of resources special frontend tools have to be used. During the insertion of HTML documents all of the necessary information is extracted by the link management to provide integrity of the links as well as additional search facilities. By monitoring all of the subsequent changes that take place within the document structure, it is possible to keep this information up to date. The access to resources in the database is accomplished by using an enhanced URL notation. A proposal for such a notation is introduced in the following section. Every time a HTML document residing in the database is requested, the Web server's URL handler has to transform the URL notation into a database query, retrieve the document from the database, use its link management to check the validity of the existing links and remove any invalid links during retrieval. If any server-side includes within the document exist, they have to be handled before sending back the document to the browser.

4.2 The URL notation for accessing information in a database system

As it has been mentioned already, using an URL notation to denote information residing in a database system has the following advantages over using special HTML tags to include database queries into HTML pages:

1. The same notation is used to access all kind of Web based resources independent of their location.

2. The notation that is used for embedding database queries into HTML pages is independent of the underlying database environment.

Thus, the proposed URL notation is capable of denoting links to multi- and hypermedia content as well as database queries including traditional data (relational or object-oriented) or multi- and hypermedia data respectively. As a result, an URL has a different semantical meaning based on the context that it is used in. In the following sections the terms *link semantics* and *query semantics* will be used to distinguish between these two semantical meanings. Based on the different semantical meanings, the link management has to perform different tasks. In case of an URL with link semantics, the link manager has to ensure the validity of the link, i. e. the link will be accessible as long as the corresponding target resource. On the other hand, an URL with query semantics remains valid as long as all of the attributes and entities exist that are part of the query in the query.

So, how is it possible for the link manager to distinguish between these two semantical meanings of the URL notation? The answer to this questions is closely related to the way Web based resources are stored in the database. Although, it is possible to store any multi- and hypermedia content as well as traditional data in the same entities, the approach that has been used here is to store the different types of resources in a centralized fashion in the so-called *resource entities*. The motivation of using these special resource entities, besides the fact that access time for these resource entities can be adapted to the particular needs by assigning them to dedicated disks and/or partioning them across several disks is illustrated in figure 2.

The application developer interacts with the system by using special frontend tools to insert, delete or modify new as well as existing entities. In the example of figure 2 it is supposed that the application developer wants to extend an existing database schema of a learning environment with two new entities, namely the 'professor' and 'lecture' entities. The 'professor' entity consists of the professor's name, his location inside of the building as well as his image. The 'lecture' entity is used to store information about lectures as its name, the day, time and location it takes place as well as the script that is used during the lecture. Thus, the appli-

⟨Database URL⟩	::=	http://⟨Host⟩[:⟨Port⟩]/⟨Path Prefix⟩/⟨Database Ref⟩
⟨Database Ref⟩	::=	⟨Database Name⟩/⟨Entity Names⟩[/⟨Choice⟩[?⟨Query Params⟩]]
⟨Entity Names⟩	::=	⟨Entity Name⟩ [& ⟨Entity Names⟩]
⟨Entity Name⟩	::=	⟨Schema⟩.⟨Entity⟩
⟨Choice⟩	::=	⟨Entity Attributes⟩ | ⟨Bytecode Ref⟩ | ⟨Presentation⟩
⟨Entity Attributes⟩	::=	⟨Entity Attribute Name⟩ [& ⟨Entity Attribute Name⟩]
⟨Entity Attribute Name⟩	::=	⟨Schema⟩.⟨Entity⟩.⟨Attribute⟩
⟨Bytecode Ref⟩	::=	⟨Entity Attribute Name⟩/⟨Class name⟩
⟨Presentation⟩	::=	⟨Presentation Template⟩'('⟨Entity Attributes⟩')'
⟨Query Params⟩	::=	⟨Query Param⟩ [& ⟨Query Params⟩]
⟨Query Param⟩	::=	⟨Entity Attribute Name⟩ ⟨Operator⟩ (⟨Entity Attribute Name⟩ | ⟨Value⟩)

Table 1: The syntax of the developed URL notation in a BNF-like syntax

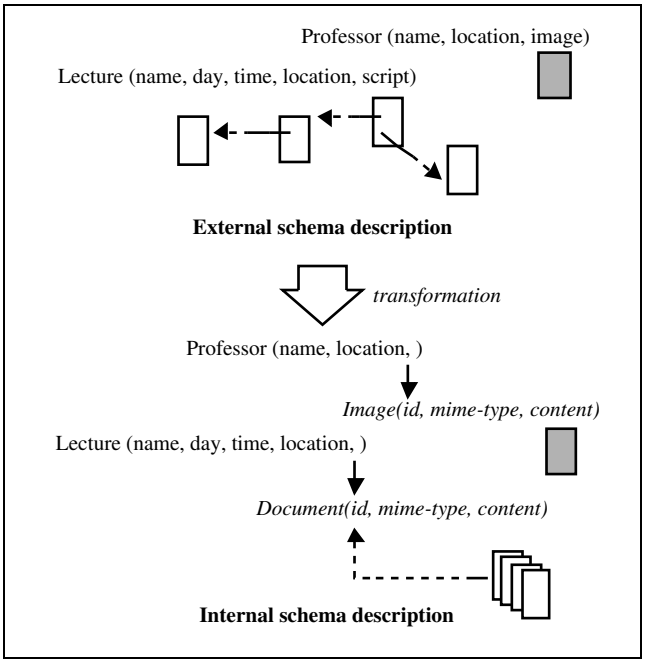

Figure 2: The notion and the usage of resource entities

cation developer interacts, defines and modifies an external schema description of the database as it is visible to the outside world. Because it is very difficult to store and handle multiple interlinked resources as an attribute value as it is needed for the script attribute of the 'lecture' entity , the system uses a different internal schema description to store this type of information. The transformation between the external and internal schema descriptions is accomplished by the appropriate frontend tools. As a consequence, multi- and hypermedia content is not stored as a part of the entity itself, but in the 'content' attribute of the appropriate resource entities which are depicted in figure 2 with an italic font. In the entity itself, the 'image' or 'script' attributes contain only references to the corresponding resource entries. Hypermedia content is further divided into its parts which in turn are stored in the appropriate resource entities with the links transformed according to the locations of their target resources as it is illustrated with the 'script' attribute.

Therefore, all of the URLs with link semantics denoting Web based resources contain only references to resource entities whereas URLs denoting database queries only contain entities that are part of the external schema description. Based on this fact, the link manager is able to distinguish between the two semantical meanings of the URL notation. Examples which are illustrating these facts more detailed follow after the proposed URL notation has been revealed.

As one can see in table 1, the proposed URL notation contains a host name (*Host*) and an optional port number (*Port*) like every other URL notation followed by a prefix (*Path Prefix*) that is used to distinguish between resources residing in the database system and in the Web server's file system. Generally, the chosen path prefix should not denote a valid filename to avoid any possible conflicts with directory names in the local file system. Examining the notation in greater detail, it is revealed how an hierarchical naming schema like an URL can be applied to database systems. All the information in a database system is partitioned into databases (*Database Ref*) which in turn consist of different entities (*Entity Names*). Entities, on the other hand, consist of attributes (*Entity attributes*). Instead of a long theoretical discourse, the usage of the URL notation will be illustrated with a few examples.

First of all, an URL with link semantics that denotes a hyperlink to a HTML document which is located on host 'www.myhost.de' in the 'education' database has the form *http://www.myhost.de/db/education/Document/content?id=5147936*. The link semantics of the URL should be obvious, because only the resource entity 'Document' is referenced in the example above. Thus, the syntax for referencing Web based resources is identical for all of the different resource types, except for Java bytecode (*Bytecode Ref*). The difference between denoting Java bytecode compared to other resources results from the way the bytecode references are handled by the Java virtual machine. Thus, the last part of an URL denoting a bytecode reference has to be a valid class name (*Class Name*). Generally, URLs denoting references to Web based resources are mostly generated automatically by the system during insertion. They can be used anywhere within a HTML document where an URL is permitted. On the other hand, a database query like "Display the name and script of all lectures that take place on Thursday" can be denoted by an URL of the form *http://www.myhost.de/db/education/lecture/name&script?day='Thursday'*. This example shows how a database query is structured. The first part of the URL denotes the entity or entities (*Entity Names*) respectively to which the query will be applied. If a query contains multiple entities, they have to be concatenated with an ampersand. The next part of the URL denotes all of the attributes (*Entity Attributes*) that form the query's result set. In addition, it is possible to restrict the query result in such a way that only instances of entities are selected that meet specified conditions (*Query Params*). By the way, these

conditions have to be specified as name/value pairs with an identical syntax as the query parameters of HTML forms. Unlike URLs with link semantics, URLs with query semantics can only be integrated into HTML pages in form of a hyperlink or can be used in the same way as a CGI script. Because of this rather limited form of embedding database queries into HTML pages, the capabilities of the database module has been enhanced to handle server-side includes of the following form

$$\langle !-- \ \#\text{include } \textit{Database URL} \ --\rangle$$

as well. This feature is also found in almost all of todays Web servers.

Sometimes it is necessary to provide a unique presentation layout for all of the information that belongs to a certain topic or to display information using a particular form of presentation. It is possible to customize the presentation of the query results using a special presentation layout (*Presentation Template*) that is stored in the database as well. Such a presentation template is simply a HTML page with variables of the form %%⟨num⟩%% which are substituted with the appropriate entities' attribute values (*Entity Attributes* in *Presentation*) during runtime. Thus, presentation templates can be used to provide a unique layout for the same type of data as well as a means to store the information that has to be presented independent of the presentation's layout. Thus, for changing a presentation's layout only the corresponding presentation template has to be modified instead of adapting multiple HTML documents according to these changes manually.

4.3 Link management

This section describes the link management in greater detail. First, the fundamental issues of link management as well as an overview of the degree of link integrity is given that the link management is able to guarantee under certain circumstances. Then, the link management's meta data structures are introduced that are necessary to store all of the relevant link information. Finally, the architecture and the functionality of the link management is presented.

4.3.1 The integrity of links

Basically, the link management provides an abstraction on top of the storage system that is used to access all of the Web based resources. Its interface consists of operations to insert, modify, delete, retrieve as well as change the location of resources. In contrast to the storage system's interface to access Web based resources, the link management extracts and maintains relevant link information during these operations to guarantee the integrity of links. Thus, in an ideal environment where all of the accesses to Web based resources are performed by using the interface of the link management, it is possible to ensure link integrity at any time. In a real Web environment, on the other hand, a lot of different tools exist that help to administrate and manage a Web site. Thus, it is a rather unrealistic asumption that all of these tools use the link management's interface to access the Web based resources. Based on this fact, the following section investigates the effect of the two most important factors with regard to the degree of link consistency that the link management is able to ensure:

The storage system: It is possible to store Web based resources either in the Web server's file system or in the database.

Location of resources: A resource can either be stored locally on the Web server's host or on a remote host somewhere on the internet.

As it has already been mentioned, a link in the World Wide Web a forward reference from a HTML document to any type of multi- or hypermedia content. Without maintaining the affiliated backward references, there is no possibility to adapt the source documents according to changes of the target resources. By using a database system for maintaining link information, the missing backward references of links can be derived. Thus, the survey in table 2 gives an overview which degree of integrity can be ensured for HTML documents that are stored on the local host. Generally, eight different cases have ot be distinguished:

Case 1-4: The links of the HTML documents that are located in the local file system cannot be automatically adapted to changes of the target resources. Thus, the link integrity is violated as long as the administrator of the local Web site is informed and changes the source documents manually. If the target resource is located in the local file system as well (case 1), it is possible to use tools like the Web crawler to analyse all of the documents that are locally available on a regular basis and identify all of the invalid links. In this case it is not possible to ensure the integrity of links too, but the period of time until invalid links are detected can be reduced. Thus, the degree of integrity that can be achieved in all of these cases is weak.

Case 5,6: Although the source documents are stored in the local database, the target resources are still located in the file system. Assuming that the link management is notified every time a target resource has been changed, the link management is able to adapt the source documents according to these changes instead of adapting the source documents manually as in case 1-4. Thus, only a weak form of link integrity can be achieved as well.

Case 7: This is the interesting case where all of the source documents as well as the target resources are located in the database. Thus, the interface of the link management is used to access all of the Web based resources and the integrity of the links can be ensured. The only problem that still may occur is due to the client side caching of resources of the Web browser and the stateless HTTP protocol. It is not possible to notify the Web browser of any changes that effect the link integrity of documents that are currently stored in the Web browser's cache.

Case 8: In the case that the target resource is stored in a remote database, it is possible to guarantee the integrity of links as well, if a protocol between the distributed link management components as described in [11] is used. The degree of link integrity that can be achieved depends on the underlying protocol that is used to propagate the changes of target resources. If the protocol uses a distributed transaction mechanism, i. e. an atomic update, to propagate these changes, it is possible to achieve a very strong degree of link integrity. On the other hand, if changes of target resources are propagated on a regular basis, i. e. a deferred update mechanism is used, only a weaker degree of link integrity can be ensured.

		Location of the target resource			
		local file system	remote file system	local database	remote database
Location of the source document	local file system	weak (case 1)	weak (case 2)	weak (case 3)	weak (case 4)
	local database	weak (case 5)	weak (case 6)	strong (case 7)	adaptable (case 8)

Table 2: Overview about the degree of link integrity that can be achieved under the different conditions.

In summary, one can say that the integrity of links can only be guaranteed in an acceptable way when all of the Web based resources are stored in the local database.

4.3.2 Meta data structures

This section gives an overview of the meta data structures that are depicted in figure 3 and have to be maintained by the link management to ensure the integrity of links. Although the meta data structures of the link management may only used be to store link information where the source documents as well as the target resources are located in the database, it should be noted that they are capable of maintaining all kind of link information independent of the resources' location. Generally, every link consists of a *source anchor* that is directly embedded into the HTML document as well as a *target anchor* which denotes the reference to a target resource.

Apart from an unique identifier, namely the *sourceId* that is used to identify the source anchor unambigiously, the *source anchor* entity consists of an *url* that denotes the source document, a reference to the affiliated target anchor (*targetId*) that constitutes the actual link as well as the location of the link within the source document, i. e. a byte range denoted by a *start*ing position and an *end* position. If the source document is located in the database, the *targetId* contains a reference to the appropriate resource entity, otherwise it is undefined.

According to the definition of the source anchor entity, the *target anchor* entity also consists of an unique identification, namely the *targetId*, as well as its location within the target resource that is denoted by a *start*ing position and an *end* position. If a target resource contains more than one target anchor, the *fragment* part of the target resource's URL is used to denote the target anchor in a unique fashion. Because a target resource may contain multiple target anchors, all of the link information that belongs to the target resource itself is included in the *resource description* entity. Thus, the resource description entity consists of an unique identifier, namely the *resDescId*, and the base *url* (without the fragment part) of the target resource. If the target resource is located in the local file system and its location has been changed, the *newUrl* indicates its new location, otherwise it is undefined. If the target resource, on the other hand, is located in the database system, the appropriate resource entity consists of a reference in form of the *resDescId* to the corresponding entry in the resource description entity.

So far, only URLs with reference semantics have been covered by the meta data structures. But, as it has been mentioned before, a hyperlink or server side include may also contain an URL with query semantics. Because the link management is able to check the validity of URLs with query semantics as well, the *query description* entity is used to store the *database*, *entities* and *attributes* that are part of the query. Additionally, the query description entity has been extended with a reference, namely the *resDescId*, to the corresponding resource description, as well, to achieve a standardized design for all types of URLs.

4.3.3 Architecture and functionality

Finally, this section describes the architecture of the database access module (see figure 1) and especially the link management in greater detail. Afterwards, the link management's interface and the usage of the meta data structures that have been described in the previous section will be revealed.

The database access module which is used to extend the functionality of the Web server as well as for the frontend tools to manage a Web site consists of the following two modules:

- The *URL handler* is responsible for transforming the URL notation into an intermediate form that is used by the link manager. Addionally, its task comprise the access to all kind of information that resides in the database as well as formatting the query results.
- The *link management* consists of a *document parser* which is responsible for extracting the URLs of the HTML documents whereas the *meta data management* is responsible for the maintenance of the meta data structures to provide link integrity. Technically, the link management is embedded into the URL handler, because it uses the intermediate URL representation of the URL handler as well as the functionality of the URL handler to access the database.

Now, that the components of the link management have been introduced, its tasks in regard of maintaining the link management's data structures which are shown in figure 3 will be described in more detail. As it has already be mentioned in section 4.3.1, the interface of the link management consists of operations to insert, modify, delete, retrieve as well as change the location of resources. Thus, the following steps have to be taken to ensure integrity of links:

- *Insertion:* Every time new resources are inserted a *resource description* with a unique *resDescId* and the corresponding *url* have to be generated. Afterwards new *target anchor*s are created containing a reference (*resDescId*) to the appropriate *resource description*, a unique *targetId* as well as the *start* - and *end* value initialized to zero. If a resource has to be inserted in the database as well, the appropriate *resource entity* is used to store the resource. As far as any HTML documents are inserted, the document parser has to extract all of the URL references as well as HTML anchor tags. For every anchor tag that is found, a new *target anchor* entry containing a reference to the HTML document, its position and the appropriate *fragment* id has to be created. For every URL that denotes a database

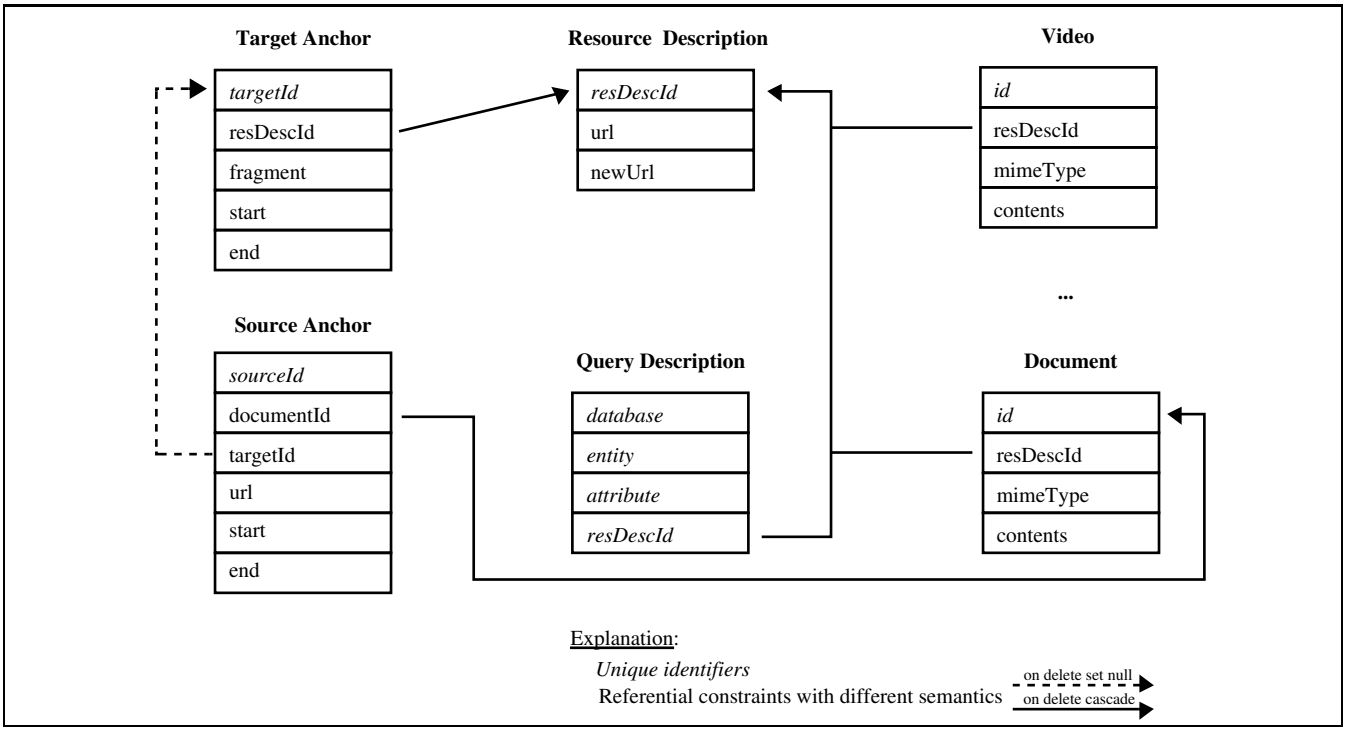

Figure 3: Meta data structures used to store all of the relevant link information

query it has to be checked if the appropriate *resource description* already exists. If not, a new *resource description*, *query description* and *target anchor* have to be inserted. For every URL that denotes a reference to a Web based resource, a new *source anchor* with a unique *sourceId*, a reference to the HTML document (*documentId*) as well as the *start* - and *end* position of the URL has to be created, while the *sourceId*s of the corresponding target anchors are stored in main memory. After all of the resources have been inserted, the link structure has to be valid. Thus, for every target anchor which is stored in main memory it has to be tested if a corresponding *target anchor* entity exists. If the test is successful, the corresponding *targetId* is stored in the appropriate *source anchor*. Otherwise, the link structure is inconsistent and all of the changes have to be undone.

- *Deletion:* Before the specified Web based resources can actually be deleted, their entries in the *resource description* entity with the appropriate URLs have to be deleted. Because of the referential integrity constraint between the *resource description* entities and the *target anchor* entities, the corresponding target anchors of these resources are removed as well. The removal of the target anchors, on the other hand, leads to the situation that the corresponding *source anchor*s have to be adapted too, because of the existing referential integrity constraint. The semantic of removing target anchors is that the corresponding links are no longer valid. This situation is reflected in the meta data structures by changing the *targetId* of the *source anchor* entity to an undefined value. Addionally, if any of the Web based resources that are about to be deleted are located in the database, they are removed from the corresponding resource entities as well, because of the referential integrity constraint between the *resource description* entity and the resource entities. In case that any of the deleted resources is a HTML document, all of its source anchors are removed from the *source anchor* entity because of the referential integrity constraint between the resource entity *Document* and the *source anchor* entity. Addionally, it is possible that the deletion of a HTML document leads to the situation that the database queries which are part of the document may be obsolete too and the corresponding *query description* entries have to be deleted. Such a situation can be discovered by counting the references from *source anchor* entries to *target anchor* entries which denote database queries. If no such references exist, the database query has become obsolete and the corresponding entry in the *target anchor* entity has to be removed. Because of the reference (*resDescId*) from the *target anchor* entry to the corresponding *resource description*, the *resource description* is removed as well. Finally, the *query description* is deleted, because of the referential integrity constraint between these two entities.

- *Modification:* In case of any modifications of resources which are not HTML documents nothing has to be done, because they do not consist of any source anchors and the target anchors are not affected by changing their content. The modification of a HTML document, on the other hand, is identical to applying a delete operation to the HTML document which is followed by an insert operation.

- *Changing location:* First, it should be noted that this operation can only be applied to resources located in the local file system, because for resources located in the database the notion of a location is not applicable.

Thus, if a resource is relocated, the *resource description* entry has to be adapted accordingly by storing its new location as the *newURL*. Additionally, a new *resource description* entry has to be created with the new location as its *url* and an undefined *newURL*. The reason for maintaining multiple resource descriptions for a relocated resource is that the source documents are not adapted automatically to the new location of their target resources. Thus, a resource description entity has to be maintained as long as it is referenced by any target anchor entry.

- *Retrieval:* Because a retrieval operation does not affect the integrity of links, only the validity of the resource's links have to be checked. Because Web based resources except for HTML documents do not contain any links, no actions have to be performed when retrieving such a Web based resource located in the database. If the Web based resource is located in the local file system, the corresponding *resource description* has to be checked if the resource has been relocated in the past. In case of a relocation, the actual *url* has to be determined by inspecting the corresponding *resource description* entries. Finally, the Web based resource has to be retrieved from the database or the local file system, respectively. On the other hand, prior to retrieving a HTML document the validity of the existing links has to be checked. Thus, all of the *targetId*s of the HTML document's *source anchor* entries containing an undefined value have to be retrieved. If such source anchor entries exist, its *start* - and *end* value have to be retrieved to remove the corresponding byte range from the HTML document during retrieval. For all of the database queries which are identified according to the reference from the document's *source anchor* to the corresponding *target anchor* and the *target anchor* to the *resource description* the existence of the corresponding *database*, *entities* and *attributes* have to be checked during runtime. This check has to be performed by retrieving the appropriate catalog information from the system catalog of the database system. If any invalid database queries exist, their position within the HTML document is determined based on the *start* - and *end* value of the corresponding *target anchor* entries and they are removed during retrieval.

Finally, it has to be added that all of the operations above are executed under transactional control. Thus, if any of the changes have to be undone, the transaction must perform a rollback, whereas committing the transaction makes all of the changes persistent.

4.4 Additional search facilities

Currently, the only possibilities to perform full-text searches within the local HTML documents are either to use one of the well-known search engines or to use tools that have been developed by oneself. On the other hand, in a Web environment that consists of a database system these queries can be performed much more efficiently using the built-in query language of the database system. So far, it is already possible to do full-text searches of HTML documents that are located in the database system by using a database query that searches the *content*s of the *Document* resource entity. But, support for efficient searches within certain relevant parts of the document like headings, captions of figures and tables as well any information that is included into the HTML document header, it is necessary to extend the functionality of the meta data management and document parser components (see figure 1) accordingly. Thus, the document parser has to extract the appropriate HTML tags during insertion or modification as well and to store them in an additional meta data structure of the following form:

Document Info (*HTMLtag*, *content*, *documentId*)

Using this kind of meta data, it is possible to efficiently search the *content* of special *HTMLtag*s in a certain document or all of the documents that are available. Just for clarification, it has to be noted that the *documentId* denotes a reference to the corresponding *Document* resource entity that contains these HTML tags. Obviously, the information in the *Document Info* entity has to be adapted according to modifications or deletions of HTML documents as well.

4.5 Navigational aid

The "lost in hyperspace" syndrome which has been observed in a lot of hypermedia environments due to a disorientation of the user after following a few hyperlinks results from a missing navigational aid in form of a map that indicates the users' current location as well as the users' path through hyperspace. As it can be seen, all of the information that is needed to generate a map of the existing document structure is already available in the meta data structure of the link management whereas the path information can be derived from analyzing the Web server's access log. By the way, the Web server's log should be stored in the database as well to be able to use the query language of the database system to obtain various access statistics. Thus, a frontend tool may use this kind of information to generate a graphical output on demand of the user to help him navigating along the existing hypermedia information.

5 Implementation

A prototype implementation of the system as it is depicted in figure 1 has been developed in a networked environment of heterogeneous Unix workstations and a relational database system. The general idea of the prototype implementation was the creation of a testbed to show the feasibility of the underlying design concepts and to build a basis for further development. In the following section the prototype implementation is revealed in greater detail:

Web-Browser: Any standard Web browser with Java support can be used to access the information residing in the database system. A number of different helper applications, like an MPEG audio and video player or a real audio player have to be installed to be able to playback audio and video clips.

Frontend tools: In the current state of the prototype implementation none of the graphical frontend tools have been developed yet. Thus, all of the resources hav to be inserted into the database using command line utilities.

Web-Server: As it has been described in section 4.1 an extensible Web server which implements the servlet interface has be chosen as the basis of the prototype implementation. Thus, the Java Web Server [4] from Sun Microsystems has been used as the software platform. The *database access module* (see figure 1) has been

implemented as a servlet that is loaded by the Web server during startup. Instead of implementing the *document parser* manually, the Java Compiler Compiler (JavaCC) [10] has been used to be able to extend the document parser according to changes in the HTML standard. Java based access to the database system which is used to store all of the meta data is accomplished by using the JDBC [6] interface.

Database system: The current implementation is based on IBM's object-relational database system DB2 Version 2.1.1 on a Sun Solaris platform. All of the multi- and hypermedia content is stored as LOB (long objects) data depending on its type either as CLOBs (character large objects) or BLOBs (binary large objects). In a further release of DB2, which has already be announced and is called DB2 Universal Database, it is will be possible to store all of the Web based resources in so-called relational extenders for text, images, audio or video respectively and to perform content based queries on multimedia data. Access to the database system is provided by DB2's native JDBC driver.

6 Conclusion

This paper presented an approach to combine the features of the World Wide Web with the functionality of database systems by storing all of the Web based resources in the database to overcome the most serious limitations of the Web. By using an URL notation to denote all of the information located in the database system, it has been demonstrated how easily references to these Web based resources as well as database queries can be embedded into HTML documents treating all kind of information in an uniform way independent of their location.

7 Future Work

The main goal is to use the prototype implementation during a practical course or a lecture to make all of the course material like scripts, programs, etc. on-line available for the students. The students' experiences will then be used for the evaluation of the prototype with respect to robustness and performance as well as missing features. Additionally, all of the frontend tools have to be developed which enable an easy administration of Web based resources. Another direction of future work is to develop an authentication model that is capable of managing Web based resources and traditional data in a more suitable manner as the World Wide Webs current authentication method that is based on protecting directory sub-trees. Thus, it would be advantageous to have a more fine-grained authentication model that is capable of granting or restricting access to single Web based resources or entity attributes.

References

[1] M. Anand, et. al. *The Web Request Broker: A Framework for Distributed Web-Based Applications.* Available at http://www.olab.com/beta/www6_1/paper.html.

[2] A. Baird-Smith. *Jigsaw Overview.* Available at http://www.w3.org/pub/WWW/Jigsaw/.

[3] J. Gaffney. *Illustra's Web DataBlade Module.* SIGMOD Record, Vol. 25, No. 1, March 1996.

[4] *The Java Server Product Family.* Available at http://jserv.javasoft.com/.

[5] B. J. Haan, et al. *IRIS Hypermedia Services.* Communications of the ACM 35(1), 1992.

[6] G. Hamilton, R. Cattell. *JDBC: A Java SQL API.* Available at http://splash.javasoft.com/jdbc/.

[7] P. Kutschera. *Combining Database Technology with the World Wide Web for Tele-Teaching Environments.* New Media for Education and Training in Computer Science, infix, 99-108, 1996.

[8] H. Maurer. *HyperWave: The Next Generation Web Solution.* Reading et. al.: Addison-Wesley, 1996.

[9] *IBM Net.Data Programming Guide.* Available at http://www.software.ibm.com/data/net.data/docs/dtwdev.htm.

[10] *Java Compiler Compiler.* Available at http://www.suntest.com/JavaCC/features.html.

[11] J. E. Pitkow, R. K. Jones. *Supporting the Web: A Distributed Hyperlink Database System.* Computer Networks and ISDN Systems 28(7-11), 981-991, 1996.

[12] W. R. Tuthill. *Don't Get Caught in the Web: A Fieldguide to Searching the Net.* Proceedings of the COMPCON'96, Santa Clara, February 25-28, 1996, 77-83, Los Alamitos, IEEE Computer Society Press, 1996.