Search Strategies for Finding Annotations and Annotated Documents: The FAST Service

Maristella Agosti and Nicola Ferro

Department of Information Engineering – University of Padua
Via Gradenigo, 6/b – 35131 Padova – Italy
{agosti, ferro}@dei.unipd.it

Abstract. This paper discusses two kinds of search strategies supported by the *Flexible Annotation Service Tool (FAST)*, an annotation service that can be used by different *Digital Library Management Systems (DLMSs)*. The first strategy concerns the search and retrieval of annotations, considered as stand-alone documents; while, the second one regards how to exploit annotations in order to search and retrieve annotated documents which are relevant for a user query. This paper describes the proposed search strategies in the light of the architectural design choices needed to support them.

1 Introduction

As observed by [10, p. 274], "the progress of the *Digital Library (DL)* field can be evaluated along several dimensions", one of which is called the *service dimension*, which "characterizes the complexity of processing that DLs and federations of DLs can manage on behalf of clients". In particular, we are interested in studying and developing a service able to add annotation capabilities on the documents managed by a *Digital Library Management System (DLMS)*, so that the service encapsulates all the complex processing needed to provide advanced annotation functionalities and can be easily "plugged" into different DLMSs.

We have designed and we are developing an annotation service for DLMSs, which is called *Flexible Annotation Service Tool (FAST)* [1, 2, 3, 4]. FAST offers basic annotation management functionalities and provides users with advanced search capabilities for retrieving both annotations and annotated documents on the basis of their annotations. This paper will introduce the search strategies supported by FAST in order to search for both annotations and annotated documents and it will describe the architecture of FAST with a particular focus on the architectural design choices which impact the search functionalities offered by the service.

The paper is organized as follows: Section 2 discusses the use of annotations in the context of DLMSs; Section 3 provides an overview of the FAST service; Section 4 describes the search strategies supported by FAST; Section 5 discusses the architecture of the system and its consequences on the offered search strategies; finally Section 6 draws some conclusions and provides an outlook of the future research work.

H.Legind Larsen et al. (Eds.): FQAS 2006, LNAI 4027, pp. 270–281, 2006.

2 DLMSs and Annotations

DLMS are currently in a state of evolution: today they are simply places where information resources can be stored and made available, whereas for tomorrow they will become an integrated part of the way the user works. For example, instead of simply downloading a paper and then working on a printed version, a user will be able to work directly with the paper by means of the tools provided by the DLMS and share their work with colleagues. This way, the user's intellectual work and the information resources provided by the DLMS can be merged together in order to constitute a single working context. Thus, the DLMS is no longer perceived as something external to the intellectual production process and neither as a mere consulting tool, but instead as an intrinsic and active part of the intellectual production process [2].

Annotations are effective means in order to enable the paradigm of interaction between users and DLMSs envisioned above, since they are very well-established practices and widely used. Annotations are not only a way of explaining and enriching an information resource with personal observations, but also a means of transmitting and sharing ideas in order to improve collaborative work practices. Thus, annotations can be geared not only to the way of working of the individual and to a method of study, but also to a way of doing research, as it happens in the Humanities. Finally, annotations allow users to naturally merge and link personal contents with the information resources provided by the DLMS so that a common context that unifies all of these contents can be created.

With this last respect, documents managed by the DLMS and annotations constitute an hypertext [3, 4], since annotations allow the creation of new relationships among existing objects, by means of links that connect annotations together with existing objects. [9] points out that annotations are one of the activities that form the basis of any collaborative effort and for which hypermedia systems are ideally suited, while [12] considers annotations as a natural way of creating and growing hypertexts that connect information resources by actively engaging users. Moreover, DLMSs do not normally have a hypertext connecting information resources with each other; thus, annotations can turn out to be an effective way of associating a hypertext to a DLMS in order to enable an active and dynamic usage of information resources. This hypertext can span and cross the boundaries of the single DLMS, if users need to interact with the information resources managed by diverse DLMSs [2]. This latter possibility is quite innovative, because offers the means for interconnecting various DLMSs in a personalized way meaningful for the end-user and, as recognized also by [10], is a big challenge for next generation DLMSs.

In this evolving context, it becomes crucial to design and develop services able to provide annotation functionalities to many different DLMSs. Moreover, besides offering various annotation management facilities, these services should pay particular attention to offer support for integrating annotations into the information access and retrieval process. Indeed, the possibilities of collaboration and active involvement with digital resources uncovered by bringing annotations into DLMSs require that annotation are an integral part of the way in which

users share, search, and retrieve information. Thus, we need to develop methods that allow us to search both for annotations themselves by taking into account their own features, and for annotated documents by exploiting the annotations linked to them. In the first case, the objective of our search are annotations and we need to develop techniques that exploit their peculiarities in order to effectively retrieve them. On the contrary, in the second case, we aim at searching annotated documents and annotations simply represent a means for retrieving better and more documents with respect to the case of a search without using annotations.

3 Overview of FAST

FAST is a flexible service designed to support both various architectural paradigms, such as *Peer-To-Peer (P2P)* or *Web Services (WS)* architectures, and a wide range of different DLMSs. The flexibility of FAST and its independence from any particular DLMS is a key feature to provide users with a uniform way of interaction with annotation functionalities, without the need of changing their annotative practices only because a user works with different DLMSs. In order to achieve the desired flexibly:

1. FAST is a stand-alone system, i.e. it is not part of any particular DLMS;
2. the core functionalities of the annotation service are separated from the functionalities needed to integrate it into different DLMSs;
3. the architecture is as modular as possible, so that different implementations of each component of FAST can be provided in order to add and/or modify its functionalities with the desired degree of granularity.

The choice of making FAST a stand-alone system is coherent with the approach adopted by different systems: for example, both Annotea [11, 13] and *Multimedia Annotation of Digital Content Over the Web (MADCOW)* [7] rely on stand-alone servers, that store and manage annotations separated from the annotated objects. On the other hand, the choice of separating the core functionalities of the annotation service, from the functionalities needed to integrate it into the different DLMSs is quite new. As a consequence of this architectural choice, it is worth pointing out that the FAST service knows everything about annotations, however it cannot do any assumption regarding the information resources provided by the DLMS, being that it needs to cooperate with different DLMSs. This situation is very different from what is commonly found today. For example, both Annotea and MADCOW are stand-alone systems but they are targeted to work with Web pages. Indeed, they assume that the annotated object has a structure compliant with *HyperText Markup Language (HTML)*, as an example, and that they can use *HyperText Transfer Protocol (HTTP)* to transport annotations. On the contrary, FAST cannot assume that it is dealing with either HTML documents or the HTTP protocol, but it has to avoid any constraints concerning both the annotated information resource and the available protocols. The only assumption about information resources that FAST can

make is that each information resource is uniquely identified by a *handle*, which is a name assigned to an information resource in order to identify and facilitate the referencing to it, such as a *Uniform Resource Identifier (URI)* or a *Digital Object Identifier (DOI)*.

FAST models annotations according to the *Entity–Relationship (ER)* schema described in [1, 4, 5]: annotations are composite and complex multi-media objects, where each part of the annotation, called *sign of annotation*, has its own medium, e.g. text or audio, and a well-defined and explicit semantics, called *meaning of annotation*, according to an ontology of agreed meanings of annotation, e.g. comment, question, and so on. Annotations can annotate multiple parts of a given *Digital Object (DO)* and can relate this annotated DO to various other DOs, if needed. Furthermore, once it has been created, an annotation is considered as a first class DO, so that it can be annotated too. In this way, the model support users in creating not only sets of annotations concerning a DO, but also threads of annotations, i.e. annotations which reply to one another. These threads of annotations are the basis for actively involving users with the system and for enabling collaboration.

From a functional point of view, FAST provides annotation management functionalities, such as creation, storage, access, and so on. Furthermore, it supports collaboration among user by introducing scopes of annotation and groups of users: annotations can be private, shared or public; if an annotation is shared, different groups of users can share it with different permissions, e.g. one group can only read the annotation while another can also modify it. Note that the annotation management engine ensure that some validity constraints are complied with: for example, a private annotation cannot be annotated by a public annotation. In such cases there is a *scope conflict* – in the example, the author of the private annotation could see both the public and the private annotation, but another user could see only the public annotation which would be annotating something hidden to this user.

4 Search Strategies Supported by FAST

As introduced in Section 1, in order to effectively exploit annotations, we need to design and develop two complementar kinds of search strategy: the first kind of search strategy is concerned with the retrieval of annotations themselves and it is presented in Section 4.1; the second kind regards how to effectively exploit annotations when we search for annotated documents; this last strategy has been firstly discussed in [3] and its key points are briefly reported in Section 4.2.

Figure 1 shows the *Unified Modeling Language (UML)* class diagram of the interfaces that describe query capabilities supported by FAST. Note that we are defining the query capabilities of FAST in term of abstract interfaces, which control the general semantics of the query; on the other hand, we have a further degree of freedom given by the possible different implementations of each interface, which can vary the functionalities actually offered, still in the broad semantics prescribed by the corresponding interface. Finally, as we will discuss in

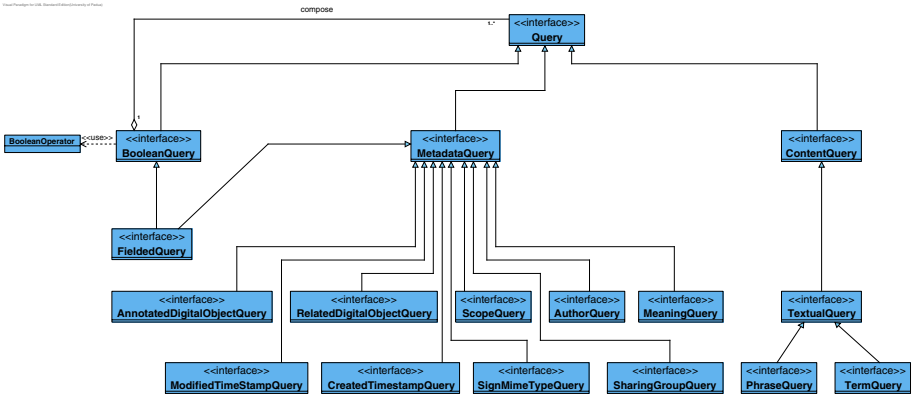

Fig. 1. UML class diagram of the query types supported by FAST

Section 5, different components of the FAST service are responsible for actually processing a query, according to its type and implementation.

4.1 Searching for Annotations

With respect to the first kind of search strategy, FAST offers three basic kinds of query: `MetadataQuery`, `ContentQuery`, and `BooleanQuery`.

The first query type, `MetadataQuery`, is intended for searching annotations on the basis of their metadata, such as for example the author of the annotation or the handle of the annotated DO. This kind of query is processed by using an *exact match* approach and annotations are retrieved in a *DataBase Management System (DBMS)* fashion. In order to actually carry out this kind of search, this interface is further specialised into a set of sub-interfaces that capture the different kinds of metadata available for an annotation. Thus, we have the following basic types of metadata query: the `AnnotatedDigitalObjectQuery` searches for all the annotations which are annotating a given DO; the `RelatedDigitalObjectQuery` has a similar purpose but for the DOs related by an annotation; the `ScopeQuery` selects annotations on the basis of their scope, i.e. private, shared, or public annotations; the `AuthorQuery` finds annotations with a given author; the `MeaningQuery` is intended for searching annotations with the specified meaning of annotation, e.g. for searching all the comments or all the counter-arguments; the `CreatedTimestampQuery` and the `ModifiedTimestampQuery` are used when we need annotations that have been, respectively, created or modified in a time stamp before, equal, or after the specified one; the `SignMimeTypeQuery` searches for annotations which contain a sign of annotation with the specified *Multipurpose Internet Mail Extensions (MIME)* type, e.g. for searching all the annotations containing a textual part or a graphical part; finally, the `SharingGroupQuery` looks annotations on the basis of the groups which are sharing them, as for example the annotations shared by a

given group with read and write permission. Note that the `MetadataQuery` covers both the metadata that are sometimes called *structural metadata*, such as the `AnnotatedDigitalObjectQuery` or the `MeaningQuery`, and those metadata that are sometimes referred as *administrative metadata*, such as `ScopeQuery` or `SharingGroupQuery`.

The second query type, `ContentQuery`, is concerned with the search and retrieval of annotations on the basis of their actual content. Thus, this kind of search requires a *best match* approach and annotations are retrieved in an *Information Retrieval System (IRS)* fashion. Even if annotations are in general multi-media compound objects, at the moment, the only medium supported by FAST for search purposes is the text. Thus, the general `ContentQuery` is specialised by the `TextualQuery` which is focused on searching for annotations that contain signs of annotations with a textual MIME type; nevertheless the query hierarchy and the architecture of FAST, as we will see in Section 5, are designed for seamlessly supporting the introduction of new content queries for other kinds of medium, such as images or audio, by simply providing new subinterfaces of `ContentQuery`. The `TextualQuery` is further specialised by two basic query types: the `TermQuery` that matches annotations against a given term, and `PhraseQuery` that matches annotations against a given phrase, that is a particular sequence of terms.

The third query type, `BooleanQuery`, represents a query that matches annotations against a combination of other queries by using the AND, OR, and NOT `BooleanOperator`s. Thus, it is a compound query whose aim is to increase the expressive power of the basic queries described above. For example, we could search for annotations with the specified author and the given scope, or we may search for annotations containing one term but not another one. Note that, even if the class hierarchy is designed for supporting the case, the present prototype implementation of FAST does not allow mixed boolean queries, that are boolean queries combining metadata and content queries together.

A final kind of query is the `FieldedQuery`, which is not a new type of query but instead it is a convenience query type. It allows us to create the most commonly used boolean queries involving annotation's metadata by ensuring that some general query semantics is preserved. For example, by using `MetadataQuery` and `BooleanOperator`, it would be possibile to create a syntactically correct query that asks for private annotations that are shared by a given group; this would result in a "wrong" query with an empty result set, because by definition private annotations cannot be shared by any group. On the other hand, the contract specified by the `FieldedQuery` interface ensures that the constraints set on the scope of an annotation are coherent with other possibile constraints on its sharing groups, otherwise an appropriate exception is thrown.

4.2 Searching for Documents by Exploiting Annotation

When we plan to exploit annotations for searching annotated documents, we need to develop a search strategy which is able to effectively take into account the multiple sources of evidence which come from both documents and annotations,

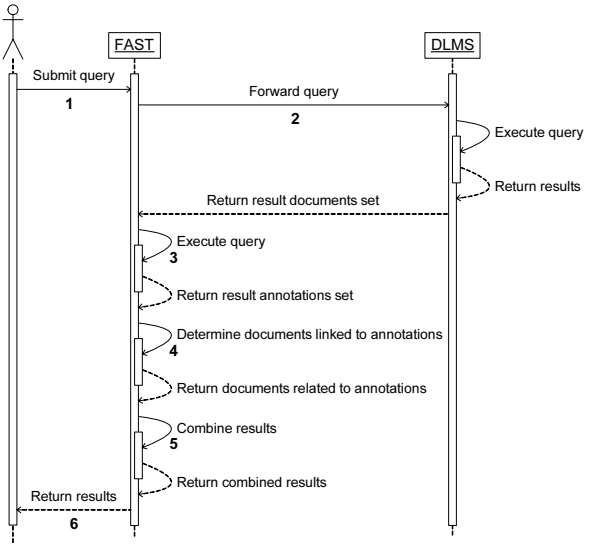

Fig. 2. UML sequence diagram of the search strategy for searching documents by exploiting annotations (from [3])

as proposed in [3]. In fact, the combining of these multiple sources of evidence can be exploited in order to improve the performances of an information management system. Our aim is to retrieve more documents that are relevant and to have them ranked in a way which is better than a system that does not makes use of annotations.

In order to carry out this search strategy, we need to deal with two kinds of DOs, that are documents and annotations. Let D be the *set of documents* and $d \in D$ is a generic document; let A be the *set of annotations* and $a \in A$ is a generic annotation; let $DO = D \cup A$ be the set of digital objects and $do \in DO$ is a generic digital object, which can be either a document or an annotation. Finally, let Q be the *set of user queries* and $q \in Q$ is a generic query. The UML sequence diagram of Figure 2 summarizes our search strategy:

1. the user submits a query $q \in Q$ to FAST;
2. FAST forwards the query to the DLMS, which searches for documents to retrieve for the query q.

   We call $R_{d,q} \subseteq D$ the result set returned by the DLMS, $s_{d,q} \in [0, 1]$ the similarity score of the document d with respect to the query q. According to our architecture, $R_{d,q}$ is completely defined and managed by the DLMS and FAST has no control over $R_{d,q}$. Thus, the DLMS has the function of providing $R_{d,q}$ and a similarity score $s_{d,q}$ for each document $d \in R_{d,q}$ to FAST;
3. FAST searches for annotations to retrieve for the query q, according to the search strategies described in Section 4.1.

We call $R_{a,q} \subseteq A$ the result set returned by FAST, $s_{a,q} \in [0,1]$ the similarity score of the annotation a with respect to the query q.According to our architecture, $R_{a,q}$ is completely defined and managed by FAST;

4. FAST determines the documents associated to the annotations contained in $R_{a,q}$, by using a *mapping function* $\mathrm{M} : A \rightarrow D$, that associates an annotation $a \in A$ to a document $d \in D$. To carry out the mapping function M, FAST exploits the hypertext existing between documents and annotations, introduced in Section 1, and follows the paths that link annotations to annotated documents.

   We call $R_{d,a} \subseteq D$ the set containing the documents associated to the annotations in $R_{a,q}$, i.e. $R_{d,a} = \mathrm{M}(R_{a,q})$; $s_{d,a} \in [0,1]$ is the similarity score of a document $d \in R_{d,a}$;
5. FAST combines the two sets $R_{d,q}$ and $R_{d,a}$ into one set $R_d = R_{d,q} \cup R_{d,a} \subseteq D$ in order to obtain only one list of retrieved documents. $s_d \in [0,1]$ is the similarity score of a document $d \in R_d$, obtained combining $s_{d,q}$ and $s_{d,a}$;
6. FAST returns the list of retrieved documents to the user.

We can point out some interesting characteristics of this search strategy. Firstly, in the fourth step FAST needs to employ both *Hypertext Information Retrieval (HIR)* [6] and data fusion techniques [8]: indeed, different paths in the hypertext allow FAST to associate annotations to documents, which are necessary to determine $R_{d,a}$ from $R_{a,q}$; furthermore, FAST has to exploit also data fusion techniques in order to compute the similarity score $s_{d,a}$ of a document d from the similarity scores $s_{a,q}$ of the annotations linked to d. Secondly, in the fifth step we need to combine the similarity scores $s_{d,q}$ computed by the DLMS with the similarity scores $s_{d,a}$ computed by FAST, which is a data fusion problem. Finally, the sequence diagram of Figure 2 highlights that we are dealing with a distributed search problem. For further details on this search strategy, please refer to [3].

With respect to the query class hierarchy shown in Figure 1, this search strategy involves mainly content queries, because we have no information about the documents managed by the DLMS and thus we remit to the DLMS the task of retrieving documents on the basis of their content. In line of principle, it would be possibile to take into consideration also some metadata query, especially on the annotation side of this search strategy, but this is left for future investigation.

5 Architecture of FAST

FAST adopts a three-layers architecture – the data, application and interface logic layers – and is designed at a high level of abstraction in terms of abstract *Application Program Interfaces (APIs)* using an *Object Oriented (OO)* approach. In this way, we can model the behaviour and the functioning of FAST without worrying about the actual implementation of each component. Different alternative implementations of each component could be provided, still keeping a coherent view of the whole architecture of the FAST service. We achieve this abstraction level by means of a set of interfaces, which define the behaviour of each

component of FAST in abstract terms. Then, a set of abstract classes partially implement the interfaces in order to define the actual behaviour common to all of the implementations of each component. Finally, the actual implementation is left to the concrete classes, inherited from the abstract ones, that fit FAST into a given architecture. Java[1] is the programming language in use for developing FAST. Java ensures us great portability across different hardware and software platforms, thus providing us with a further level of flexibility.

The UML class diagram of figure 3 presents the main interfaces involved in the definition of the data and application logic layers, which are the ones responsible for the actual query processing. The user interface layer is not reported in figure 3 since issues concerning the visualization of query results are out of the scope of the present paper.

5.1 Data Logic Layer

The data logic layer manages the actual storage of the annotations and provides a persistence layer for storing and retrieving the objects which represent the annotation and which are used by the upper layers of the architecture.

`Datastore` is a façade for the following interfaces: `AnnotationDAO`, `UserDAO`, `GroupDAO`, `MeaningDAO`, and `LoggerDAO`, which define the operations needed to ensure the persistence of the different objects managed by the system. These interfaces are designed according to the *Data Access Object (DAO)* design pattern[2]. The DAO implements the access mechanism required to work with the underlying data source, e.g. it may offer access to a *Relational DBMS (RDBMS)* by using the *Java DataBase Connectivity (JDBC)*[3] technology. Besides ensuring the persistence of the differen objects managed by the system, the `Datastore` is responsible also for actually processing both metadata queries and boolean queries, composed by only metadata queries. We use the PostgreSQL[4] DBMS in order to perform the actual storage of the annotations, and a concrete class implementing the `Datastore` interface has been developed in order communicate with PostegreSQL by using JDBC.

The `Indexer` interface is responsible for the processing of content queries. In particular, in the prototype of FAST we have implemented a textual indexer based on the Lucene[5] library able to manage the different kinds of textual query. As introduced in Section 4.1, support for further kind of media can be easily added by proving additional implementations of the `Indexer` interface, specialised for the desired medium.

Finally, the `Datalogic` interface provides coherent access to the underlying components and forwards queries either to the `Datastore` or the the proper `Indexer`, according to whether they are metadata or content queries.

[1] `http://java.sun.com/`
[2] `http://java.sun.com/blueprints/corej2eepatterns/Patterns/`
[3] `http://java.sun.com/products/jdbc/`
[4] `http://www.postgresql.org/`
[5] `http://lucene.apache.org/`

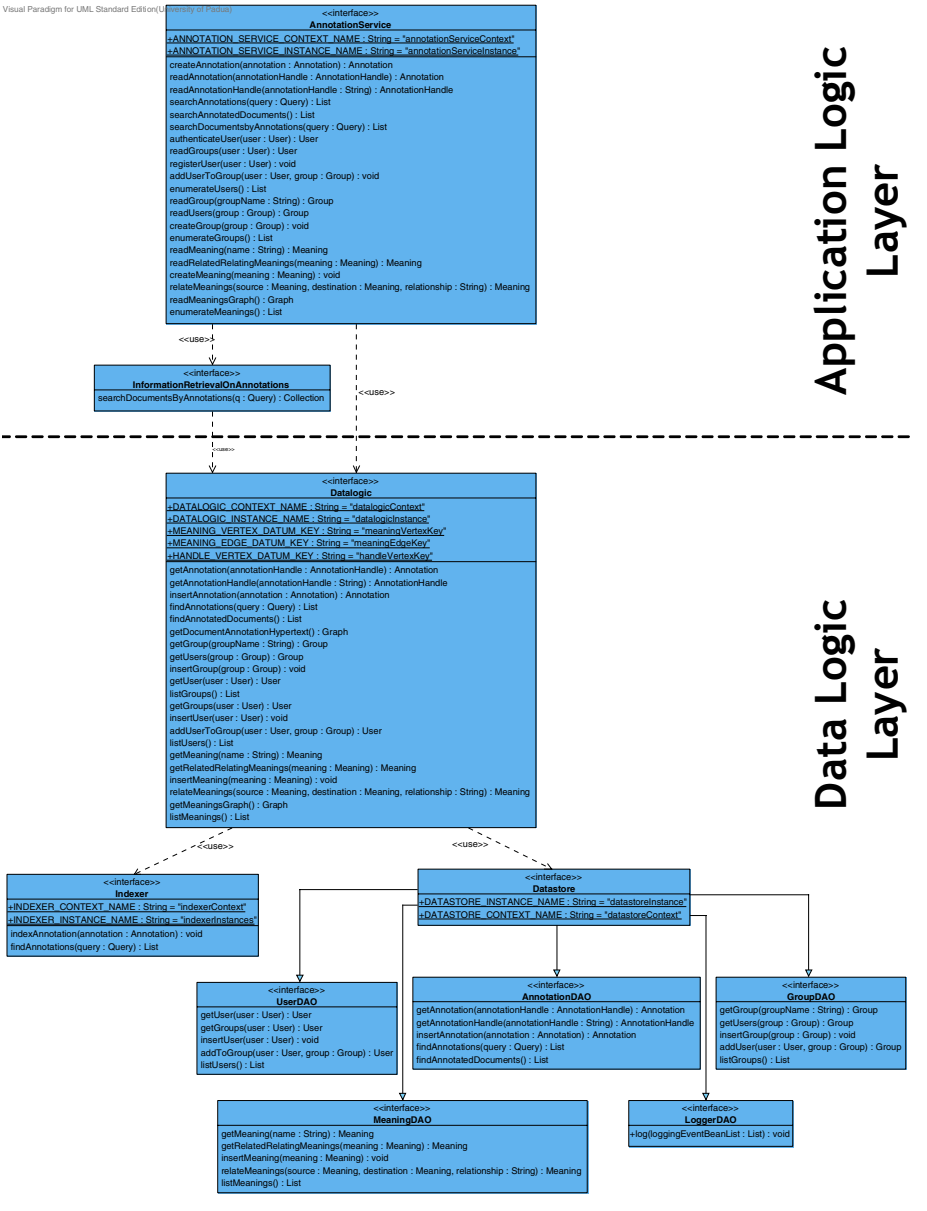

Fig. 3. UML class diagram of the data and application logic layers of FAST

5.2 Application Logic Layer

The application logic layer provides advanced functionalities that make use of annotations, such as for example the search strategy described in Section 4.2. As in the case of the data logic layer, we define a set of abstract API that make

the access to the FAST service functionalities independent from the particular implementation provided.

In particular, the `InformationRetrievalOnAnnotations` interface is responsible for carrying out the search strategy introduced in Section 4.2. This interface is currently being implemented and relies on the `Datalogic` for carrying out the part of the search strategy that involves searches on annotations.

Finally, the `AnnotationService` interface provides coherent access to the underlying components and mainly forwards the requests it receives to the right component, after having analyzed them.

6 Conclusions

We have introduced the issues related to the effective searching of both annotations and annotated documents, when we plan to offer an annotation service that supports both the collaboration and the active involvement of users in a DLMS. Moreover, we have presented our prototype of annotation service, called FAST, introducing its architectural features with respect to the different supported search strategies.

Future research will concern the study of more complex query processing algorithms able to support mixed boolean queries, combining metatada and content queries at the same time. Furthermore, we will need to evaluate the retrieval performances of the proposed algorithms by using standard information retrieval methodologies. Finally, there is a lack of experimental test collections with annotated digital contents. Thus, the future research work may also concern the design and development of this kind of test collection.

Acknowledgements

The work reported in this paper has been partially funded by Italian Ministry of Education (MIUR) under the Project of Relevant National Interest (PRIN) called "Methods for a digital corpus of illuminated manuscripts" (ISA). The work was also partially supported by the DELOS Network of Excellence on Digital Libraries, as part of the Information Society Technologies (IST) Program of the European Commission (Contract G038-507618).

References

1. M. Agosti and N. Ferro. Annotations: Enriching a Digital Library. In T. Koch and I. T. Sølvberg, editors, *Proc. 7th European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2003)*, pages 88–100. LNCS 2769, Springer, Heidelberg, Germany, 2003.
2. M. Agosti and N. Ferro. A System Architecture as a Support to a Flexible Annotation Service. In C. Türker, M. Agosti, and H.-J. Schek, editors, *Peer-to-Peer, Grid, and Service-Orientation in Digital Library Architectures: 6th Thematic Workshop of the EU Network of Excellence DELOS. Revised Selected Papers*, pages 147–166. LNCS 3664, Springer, Heidelberg, Germany, 2005.

3. M. Agosti and N. Ferro. Annotations as Context for Searching Documents. In F. Crestani and I. Ruthven, editors, *Proc. 5th International Conference on Conceptions of Library and Information Science – Context: nature, impact and role*, pages 155–170. LNCS 3507, Springer, Heidelberg, Germany, 2005.
4. M. Agosti, N. Ferro, I. Frommholz, and U. Thiel. Annotations in Digital Libraries and Collaboratories – Facets, Models and Usage. In R. Heery and L. Lyon, editors, *Proc. 8th European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2004)*, pages 244–255. LNCS 3232, Springer, Heidelberg, Germany, 2004.
5. M. Agosti, N. Ferro, and N. Orio. Annotating Illuminated Manuscripts: an Effective Tool for Research and Education. In M. Marlino, T. Sumner, and F. Shipman, editors, *Proc. 5th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL 2005)*, pages 121–130. ACM Press, New York, USA, 2005.
6. M. Agosti and A. Smeaton, editors. *Information Retrieval and Hypertext*. Kluwer Academic Publishers, Norwell (MA), USA, 1996.
7. P. Bottoni, R. Civica, S. Levialdi, L. Orso, E. Panizzi, and R. Trinchese. MADCOW: a Multimedia Digital Annotation System. In M. F. Costabile, editor, *Proc. Working Conference on Advanced Visual Interfaces (AVI 2004)*, pages 55–62. ACM Press, New York, USA, 2004.
8. W. B. Croft. Combining Approaches to Information Retrieval. In W. B. Croft, editor, *Advances in Information Retrieval: Recent Research from the Center for Intelligent Information Retrieval*, pages 1–36. Kluwer Academic Publishers, Norwell (MA), USA, 2000.
9. F. G. Halasz. Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems. *Communications of the ACM (CACM)*, 31(7):836–852, 1988.
10. Y. Ioannidis, D. Maier, S. Abiteboul, P. Buneman, S. Davidson, E. A. Fox, A. Halevy, C. Knoblock, F. Rabitti, H.-J. Schek, and G. Weikum. Digital library information-technology infrastructures. *International Journal on Digital Libraries*, 5(4):266–274, 2005.
11. J. Kahan and M.-R. Koivunen. Annotea: an open RDF infrastructure for shared Web annotations. In V. Y. Shen, N. Saito, M. R. Lyu, and M. E. Zurko, editors, *Proc. 10th International Conference on World Wide Web (WWW 2001)*, pages 623–632. ACM Press, New York, USA, 2001.
12. C. C. Marshall. Toward an Ecology of Hypertext Annotation. In R. Akscyn, editor, *Proc. 9th ACM Conference on Hypertext and Hypermedia (HT 1998): links, objects, time and space-structure in hypermedia systems*, pages 40–49. ACM Press, New York, USA, 1998.
13. W3C. Annotea Project. `http://www.w3.org/2001/Annotea/`, October 2005.