

Structured Peer-to-Peer Search to build a Bibliographic Paper Recommendation System

By
Pleng Chirawatkul

Supervised by
Prof. Dr.-Ing. Gerhard Weikum
Sebastian Michel
Matthias Bender

A thesis submitted in partial fulfillment of the requirement
for the degree of Master of Science

Computer Science Department
Saarland University

October, 2006

Abstract

The MINERVA project is a distributed search engine prototype based on a Peer-to-Peer architecture. The MINERVA provides a functionality to search for documents from the selected top-k peers who have the most possibility to contain documents corresponding to the user specified query. In some cases, users may want to know information about the document such as bibliographic information or user recommendation in order to justify the value of the document before they really reach the document. In this thesis, we study about extending the MINERVA system to support the bibliographic information search or recommendation search in the shape of attribute value based search. In our design, we specify each bibliographic information entry or recommendation entry as an attribute value of the document. We propose three designs to modify the system's global directory to support attribute value based search. We also implement the other extension parts of the application such as document database as well as new GUI to support new features. In our extended MINERVA system, users are allowed to search for documents from the bibliographic information, recommendation, or any attribute values. Users are allowed to retrieve attribute values corresponding to the specific document, and also allowed to insert and distribute the bibliographic information, recommendations or any attribute value to documents.

I hereby declare that this thesis is entirely my own work and that I have not used any other media than the ones mentioned in the thesis.

Saarbruecken, October 30, 2006

Pleng Chirawatkul

Acknowledgment

I would like to thank my academic advisor Professor Gerhard Weikum for his guidance and encouragement through the all my master study. I would like to thank my supervisor Sebastian Michel, Matthias Bender for their assistance, and all supporting during my master thesis. I would like to thank my parents for everything they did for me. And finally, I want to thank to Chamairat Samorn for her encouragement all the time through my thesis.

Contents

1. Introduction	1
1.1. Motivation	1
1.2. Contribution	2
1.3. Thesis overview	3
2. Peer-to-Peer System	4
2.1. Peer-to-Peer Basics	4
2.2. Pastry	6
2.3. Past	7
2.4. Related Work	9
3. MINERVA	12
3.1. System Design	12
4. System design	16
4.1. Use Cases	16
4.2. Partial Document Index Approach (PDI)	17
4.3. Document List Approach (DL)	21
4.4. Full Document Index Approach (FDI)	25
4.5. Discussion and Decision	28

5. Extending MINERVA	31
5.1. Global Index Extension	31
5.2. Database Extension	33
5.3. Query Processor Extension	34
5.4. Recommendation Input System	36
6. Implementation	39
6.1. Class and Methods Involve in Pastry Connection	39
6.2. Classes and Methods Involve in Global Index Generation	40
6.3. Classes and Methods Involve in Query Processing	44
6.4. Methods Involve in Database Connection	47
6.5. GUI	48
6.6. Problems	55
7. Conclusion and Future Work	56
7.1. Conclusion	56
7.2. Future Work	57

List of Figures

2.1	Distributed Hash table	5
2.2	Routing table of a Pastry node with nodeId 64b4x, where $b = 4$ and x denote an arbitrary suffix.	7
2.3	Software architecture of a peer node	10
3.1	Database schema of the MINERVA System	13
3.2	The MINERVA System	13
3.3	P5 retrieves peer lists for each query term from the global directory	14
3.4	Peer forward query to selected peers	15
4.1	Example of posts created in the PDI of peer <Peer A>	19
4.2	Example query process for query “quality=good”	20
4.3	Example query process when user wants to retrieve more attribute values for document	21
4.4	Example of posts created in PDI for peer <Peer A>	24
4.5	Query step 4-5 for DL approach	24
4.6	Query step 6 for DL	25
4.7	Example of posts created in FDI for peer <Peer A>	27
4.8	Query steps for FDI	28
5.1	Extended database schema in MINERVA	33
5.2	Dataflow diagram of an extension part to generate AVPosts and DocIdPosts	34
5.3	Dataflow diagram of query processing of extended MINERVA	37

5.4	Dataflow diagram of recommendation input part	38
6.1	Dependency diagram of class CreateJoin	40
6.2	Flow chart of MinervaPastContent.checkInsert	42
6.3	UML class diagram related on posts	43
6.4	UML class diagram related in query processing	46
6.5	Example query in MINERVA GUI	49
6.6	Main GUI of MINERVA and local peer's information windows	50
6.7	Login dialog	50
6.8	Option dialog	51
6.9	Query result reporting frame	51
6.10	Attribute input frame (Manual)	52
6.11	GUI, attribute input frame linking from query result frame	52
6.12	UML class diagram on GUI of MINERVA	53
6.13	UML Class diagram of GUI involve in query input and output	54

Chapter 1

Introduction

1.1 Motivation

This thesis is a part of the Peer-to-Peer Web search engine project developed in the Database and Information System department in the Max-Planck Institute for Computer Science. MINERVA [11] is the search engine that provides features for user to retrieve documents for given terms. Each peer in the MINERVA system maintains a database containing statistical information that crawled from their own collection or the WWW by an external crawler or indexer. This statistical information is used to create summary information (Metadata) of the peer who has the document for each term index. Metadata is distributed to the global directory to form a global index that map from term to a list of peers who have documents corresponding to the term along with statistical information about the term. The query routing process is first query for a metadata in the global directory and route the query to the selected peer.

However in the current implementation of the MINERVA system, information such as bibliographic information of the document cannot be used in the query process. Bibliographic information as well as document recommendations can be treated as the attribute of the document. There are

some exist works such as [2] and [13] that can be used to improve the searching feature of MINERVA to support attribute searching.

In this thesis, we are interested in building an extension to the MINERVA system to support attribute value based search. Three use cases need to be fulfilled. The first case is that the user may wants to retrieve related documents fro a specific attribute. The second use case is to allow user to create and distributed attributes for a specific document. The last use case is that we are interested in retrieving document attributes for a particular document.

1.2 Contribution

The extension of the global directory is the most important part of our work. To support all features mention in the last section, we design and implement additional index that map from attribute of the document to the document information to support attribute query. The other type of the global index that we had implemented is the index that maps from document to attributes. This type of index is very useful when the user wants to get information about a specific document as well as recommendations of the document. Many operations are added to integrate the attribute searching part with the term searching part in the current version of MINERVA, to provide feature to make a query that consist of terms along with attributes if the document.

We implemented a new GUI for the application to support all additional features. The new GUI allows user to execute a query with term and attribute, allows users to choose difference merging technique, and also allow user to add attributes to documents. Furthermore, we have completed the migration from the Chord overlay network to the Pastry overlay network.

1.3 Thesis overview

Chapter 2 is an overview of the Peer-to-Peer system, a summary description of the Pastry overlay network and Past storage utility, examples of existing peer-to-peer search engine, and a summary description of other works related to our work. In the chapter 3 we describe the system design as well as the query execution implemented in the MINERVA system. Chapter 4 presents use cases and our system design to support use cases of the system. In the chapter 5 we describe the extensions of MINERVA that are needed to support our approach. In the chapter 6 we describe the implementation detail of the application as well as problems that occur during implementing. In chapter we conclude our thesis and present a future work proposal.

Chapter 2

Peer-to-Peer System

2.1 Peer-to-Peer Basics

A Peer-to-Peer system is a distributed computing system in which each peer (user) in the system can communicate (or share resource) directly with each other peer in the system. One definition of a peer-to-peer system is given in [18]: “A distributed network architecture may be called a peer-to-peer (P-to-P, P2P ...) network, if the participants share a part of their own hardware resources (processing power, storage capacity, printer ...). These shared resources are necessary to provide the Service and content offered by the network (e.g. file sharing or shared workspace for collaboration). They are accessible by other peers directly, without passing intermediary entities.”

Early popular peer-to-peer system started around 1990 initially with the instance messaging service such as ICQ. The other famous peer-to-peer applications are file sharing applications such as Napster [21] and Gnutella [ww]. Napster is said to be a Hybrid peer-to-peer system according to the definition given in [18]. In Napster, indexes to file located in the user peer are stored in the centralized server. User has to retrieve index to the file from the central server and then retrieve the file directly from the peer who own it. The

main disadvantages of Napster are about scalability and single point of failure issue due to the centralized index server. In Gnutella, the query is flood from the query to all its neighbors and then forward to their neighbors again until the file is found or the time-to-live of the query has expired. In this approach, the query result may never be retrieved from far away peers and flooding consumes a lot of network bandwidth. These early systems are called “Unstructured P2P Systems” [7]. The modern Peer-to-Peer System such as Chord [24], CAN [25], Pastry [3], and Tapestry are based on Distributed Hash Table (DHT) [8], “Structured P2P System” [7]. In the DHT based Peer-to-Peer system, each node is responsible for a part of global directory, i.e. responsible for a key range on the identifier space. The data is stored in the global directory associated with the key of the data. All nodes in the system can put or get files from the system using the functions provided by the DHT. Figure 2.1 represents the distributed hash table. In the following section we give a short introduction to Pastry.

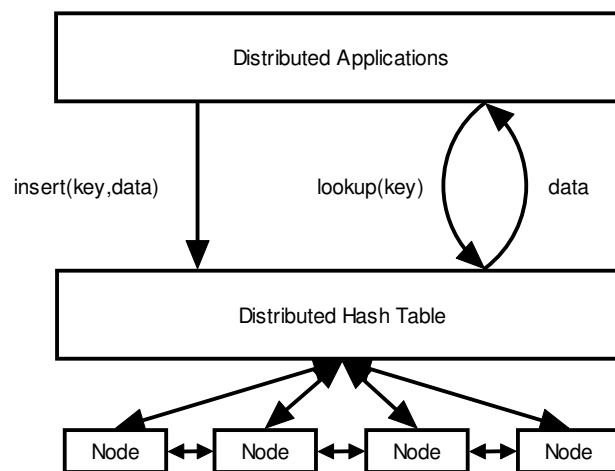


Figure 2.1: Distributed Hash table [23]

2.2 Pastry

Pastry [3] is a scalable and self-organizing peer-to-peer overlay network. Each pastry node is identified by a unique 128-bits identifier in a circular nodeId space in form of a sequence of 2^b based-digits. The nodeId is given on joining to the system randomly or generated from a hash of the node's public key or IP address.

Each pastry node maintains a routing table, a neighborhood set and leaf set. A routing table is organized into $\lceil \log_c N \rceil$ rows where $c = 2^b$ and $2^b - 1$ columns (b is a configuration parameter with typical value of 4 and N denote the number of Pastry node in the network), figure 2.2. Each row contains the IP address of nodes those nodeIds share the same prefix (n digits) with the local node and the entry left empty if there is no appropriate nodeId prefix is known. Leaf set is a set of l nodes with nodeIds $l/2$ numerically closest larger and $l/2$ numerically closest smaller with the local nodeId (typical value of l is approximately $\lceil 8 * \log_{16} N \rceil$). The neighborhood set of nodeId and IP address of $|M|$ closest nodes with the local nodeId. The neighborhood set is usually not used in routing the message.

On routing the message, the node first looks in its leaf set if the key is in the range covered by the leaf set. Then the node forwards message directly to the node with nodeId closest to the key. But if the key is not covered by the leaf set, the node then search in its routing table for a node that nodeId share at least one more common prefix with the key than the current node and numerically closer to the key than the current nodeId. If is no node is found, the message is forwarded to the node that nodeId share a common prefix with the key as long as the current node, but is numerically closer to the key. Such that a node is exist in the leaf set or the local node is numerically closest to the key.

0	1	2	3	4	5		7	8	9	a	b	c	d	e	f
x	x	x	x	x	x		x	x	x	x	x	x	x	x	x
6	6	6	6		6	6	6	6	6	6	6	6	6	6	6
0	1	2	3		5	6	7	8	9	a	b	c	d	e	f
x	x	x	x		x	x	x	x	x	x	x	x	x	x	x
6	6	6	6	6	6	6	6	6	6	6		6	6	6	6
4	4	4	4	4	4	4	4	4	4	4		4	4	4	4
0	1	2	3	4	5	6	7	8	9	a		c	d	e	f
x	x	x	x	x	x	x	x	x	x	x		x	x	x	x
6	6	6	6		6	6	6	6	6	6	6	6	6	6	6
4	4	4	4		4	4	4	4	4	4	4	4	4	4	4
b	b	b	B		b	b	b	B	b	b	b	b	b	b	b
0	4	2	3		5	6	7	8	9	a	b	c	d	e	f
x	x	x	x		x	x	x	x	x	x	x	x	x	x	x

Figure 2.2: Routing table of a Pastry node with nodeId 64b4x, where $b = 4$ and x denote an arbitrary suffix.

Due to only $\lceil \log_c N \rceil$ where $c = 2^b$ levels are populated in the routing table and the set of nodes that share the common nodeIds with the key longer than the current node is reduced by 2^b in each step, hence Pastry can normally route a message to the numerically closest node within $\lceil \log_c N \rceil$ where $c = 2^b$ step. If the key is covered by a leaf set, it takes only one hop to the numerically closest node. If the key is not covered by the leaf set and the node with appropriate prefix is not found in the routing table, due to very small probability of this case to occur, no more than one additional routing step results with high probability [3].

2.3 PAST

PAST [4, 5] is a large-scale peer-to-peer persistent storage utility on top of Pastry. The PAST system composes of nodes connected on the overlay network and each node is able to request for inserting or retrieving a file. Each node in PAST is assigned 128-bit identifier (nodeId), computed from hash (e.g., SHA-1) of node's public key.

When the file is inserted to PAST, the file is assigned a 160-bit identifier (fileId), computed from the file's name, the owner's public key, and a randomly chosen salt. Then copies of the file are routed by Pastry to the k (k is a replication factor that depends on the availability and may vary among files) nodes that nodeIds are numerically closest to the 128 most significant bits of the fileId. This make the file available if there is at least one of k node is alive and reachable. When the insert request message reaches the node that nodeId is k numerically closest, the node first checks if the copy of the file can be stored on the local disk. If the file can be stored, it stores the copy of the file on the local disk, returns a store recipe, and forwards the message to the other nodes that have $k - 1$ numerically closest nodeId.

If the file cannot be stored on the local disk, PAST provides feature to divert the copy of the file to the other node, in order to balance the load among nodes with different storage size. When the node cannot stored the file, it looks in its leaf set for a node that is not in the k closest and does not already store a diverted copy of a file then send the copy to be stored in that node on it behalf and then creates a pointer to the node that stored the copy and return a stored recipe. But if the copy of the file cannot be diverted to any node its leaf set, the negative acknowledgment is return back to the client node in order to inform inserting failure. Then the client node tries to generate a new fileId from the different salt and inserts the file again. The client node is limited to retry an insert operation up to three times, if that last try fail, the operation is aborted and the failure report is sent to the application. The application may need to resize the file (e.g. by fragmenting) or reduce a replication factor (k).

PAST also provide caching feature for popular files. For example, if the file is popular in some local clusters, it is better to have the copy of the file cache near these clusters, this aims to improve system performance and increasing the storage utilization. PAST nodes use the unused storage space to cache files. These cached files can be added or removed at any time such as

when the storage space is needed for storing the primary or diverted copy of the file.

2.4 Related Work

Jun Gao et. al. propose the music information retrieval system based on the music file description in [1] and [2]. They present an application to search for music files in a peer-to-peer system that support information retrieval based on content and Music File Description (MFD). Music File Descriptions consist of attribute value pairs (AVPairs), $MFD : \{a_1 = v_1, \dots, a_n = v_n\}$ where $a_i, i = 1 \dots n$, that annotated the music file such as $\{composer = Mozart\}$. Some MFD such as artist, composer, or album can be specified manually by the user, some MFD such as Spectral Centroid can be automatically generated by the Music Feature Extraction Engine.

In the system, MFD is registered to the Content Discovery System (CDS) that run top of DHT-based peer-to-peer overlay network. In CDS, each AVPair in MFD is applied with a uniform hash function such as SHA-1 individually to obtain an identifier (key). Then the MFD is sent to be stored on the local database (MFD Database) of a node that is responsible for this key range. Hence, instead of maintaining only a music file database, nodes also have to maintain MFD databases that store information about the location of music files related to AVPairs responsible.

The search operation is divided into two types. User want search for a song that exactly matches with the query such as $Q : \{a_1 = v_1, \dots, a_n = v_n\}$. Each AVPair in Q is the hashed separately to obtain a key $k_i = H(a_i = v_i)$ where $i = 1 \dots n$ and then send each single query to the peer that responsible for each key range. When the peer receives the query, it makes a pair-wise comparison with all entries stored in its own MFD database for a matching MFD. The second is a similarity search, in this type of query; user

may come up with some feature extract from music file and want to search for the other songs that are similar to this song. The query is processed as done in the exact search, when the query arrive the peer responsible for the key range, instead of making a pair-wise matching, distance measure techniques such as Cosine distance or Manhattan distance is applied to compute the similarity score between query vector and MFD stored on local database. The score is then ranked and return the top-k similarity to the user.

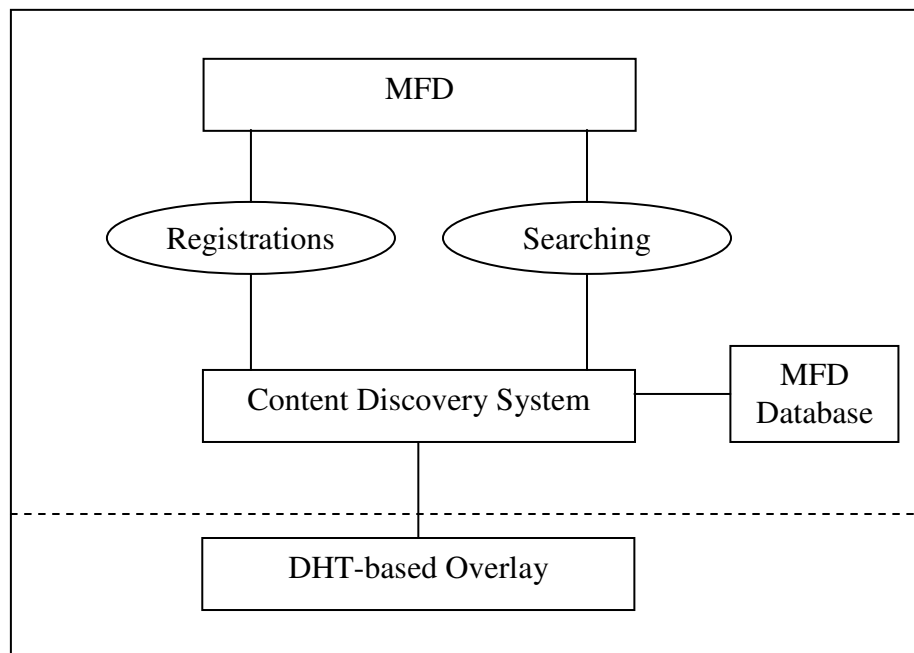


Figure 2.3: Software architecture of a peer node. [1]

Bibster [6] is a Peer-to-Peer system for exchanging bibliographic data. In this work, the authors are making the assumption that many researchers are keeping lists of bibliographic metadata preferably in the BibTex format. The bibliographic metadata is imported to the system by the user to the local knowledge repository. BibTex file is transformed and automatically advertised in two common ontology based representation, the Semantic Web Research Community (SWRC), and the ACM Topic Hierarchy. SWRC describes generic

aspects of the bibliographic metadata and ACM Topic Hierarchy is used to describe the expertise (a semantic description of the local node repository) of the peer.

Querying can be formulated in two ontologies. For the SWRC the query may concern on attribute such as author, type (Using terms from the SWRC [6]). The second type of the query concern the specific Computer Science term (using the ACM Topic Hierarchy [6]). Upon receiving a query from the user, the peer first evaluates against a local repository and then decides which peer has the highest probability to contain a query result. In the peer selection process, query subject (abstraction of query expressed in common ontology) is applied with the similarity function against peers' expertise based on a shared ontology. The query is send to Top-k highest similarity score peers and then the query result is returned.

Chapter 3

MINERVA

In this chapter we present a MINERVA project that our application is based on. The MINERVA Project [11] is a peer-to-peer search engine built on Pastry overlay network.

3.1 System Design

The MINERVA system is formed by peers joining in a Pastry overlay network. Every peer in the system is identified by the `nodeId` and responsible for a part of global directory based on top of DHT. Each peer maintains a local index in a local database containing statistical information that crawled from their own collections or the WWW by an external crawler or indexer. Figure 3.1 shows a database schema that is used in the MINERVA system. Each peer creates and publishes a metadata (post) for each term t_i in its local index containing contact information of the peer and statistics to calculate IR-style measure (peer list) to the global directory. Each post has an identifier calculated from hash of term $Id = hash(t_i)$, and is distributed in the system to form a global index (global directory). The global index in the MINERVA system contains indexes that map from term to peer list that formed by posts distributed from all peers over the network.

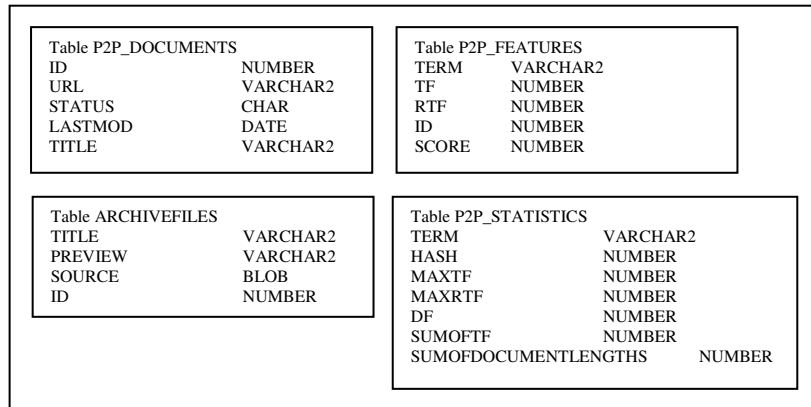


Figure 3.1: Database schema of the MINERVA System

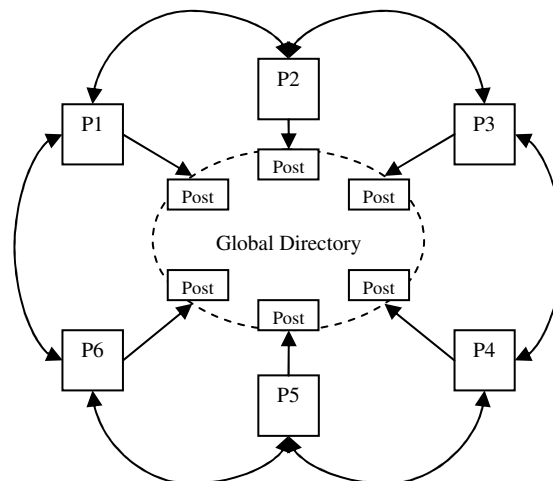


Figure 3.2: The MINERVA System

In the query process, the system splits a multi-term query into many single-term queries, stems every query term, and performs a query in the global directory for metadata (posts) related to each query term. Each query term t_i is assigned with the identifier calculated from a hash of term $Id = hash(t_i)$. The function $lookup(Id)$, provided by the distributed hash table in an overlay network, returns a peer p_i who responsible for this key range. The peer list

that contains contact information for each t_i is retrieved from the global directory by sending a peer list request to the responsible peer p_i . Each query term has to perform the global query separately, result in a set of PeerList, each relates to each query term. Figure 3.3 presents global query execution.

The routing process is beginning with the peer selection method and then route the query to the selected peers. The peer selection starts with merging all peer list retrieved from the global directory to a single peer list base on score calculated from statistical information in the retrieved peer lists [11]. The merged peer list is ranked with respect to score to obtain the most relevant k peers for the complete query execution. The system performs a complete query execution by forwarding the query to the selected peers to be executed locally on their own local indexes. Each selected peer results in the top-k document list for the query received. Result lists are retrieved from peers and then finally are merged into a single top-k document list.

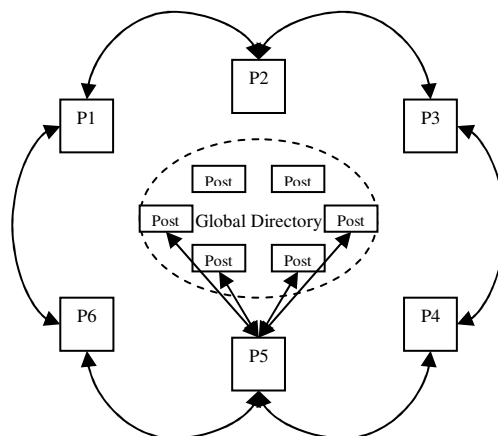


Figure 3.3: P5 retrieves peer lists from all query terms from the global directory

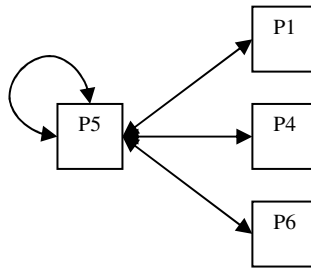


Figure 3.4: Peer forward query to selected peers

Bibliographic information can be treated as an attribute of the document such as “year=2006”. In the MINERVA system user can making a query with only term. But in many cases user may want to make a query for document by s attributes such as “year=2005”. The current implementation of the MINERVA system cannot answer this type of query. Another useful application related to attribute retrieval is the document recommendation system. Similar to the bibliographic information, document recommendations can be treated as attributes of the documents such as “quality=good”. This application is very useful when users want to find out that the retrieved document is value to read or not, or users want to recommend the document to the other users in the system.

In this thesis, we are interested in extending the MINERVA system to support bibliographic search as well as recommendation system in the shape of attribute value based search. The later chapter in this thesis will present designs and implementation of the system.

Chapter 4

System Design

4.1 Use Cases

In the current implementation of the MINERVA system, the search functionality is limited to a simple term based query. User cannot make a query about attribute values of the document. In our application, we are interesting in three use cases. The first use case is to execute a query with attribute values of the document (Attribute value to documents, AD), the second use case is allowing user to insert and distribute attribute values of the document to the system (Attribute Value Creation, AC), and the last use case is to allow user to searching for attribute values for the specific document created by other users in the network (Document to attribute values, DA).

The first use case, attribute values to document (AD), is to retrieve related documents from the specific AVPairs. User wants to retrieve documents or publications by executing queries with terms, recommendation level or other attribute values. For example, user may wants to search for publications that contain term “LSI” having recommendation level “excellent” and is published in year 2003. In order to support this case, in our global

directory, we must provide indexes that map attribute to document and other information that is useful to retrieved document from the specified keyword.

The second use case is called attribute creation (AC). After a user read a document, the user may wants to create a recommendation about the document, or may wants to add attribute values for the document such as information about the author of the document, the year of publication, or other bibliographic information and distribute it to the system. To support AC, our application should provide features to receive attribute values for a particular document from the user, store received information in the local database and distribute the information to the system.

The last use case is called document to attribute values (DA). Here the user wants to retrieve document attribute values for a particular document. For instance, user may want to know information such as a recommendation level, the author of the document or the year of publication for a particular document that is distributed in the system. The global directory should contain indexes that map document (identifier) to attribute values. To support three use cases mentioned, three different designs is propose to extend the MINERVA system.

4.2 Partial Document Index Approach (PDI)

Recommendation or any attribute value of document can be processed as attributes of the document. In the current MINERVA System, the global directory contains indexes that map from term to peerlist. To support use cases mention in the last section two types of index should be added in the system. The first type is index that map from attribute to document. Attribute of the document is created manually by the user in the form of attribute and attribute value pair (AVPair). On receiving the query with AVPair such as “quality=good”, the global index should return all documents or contact information of peers who have information about this document. To form this

type of index, each peer in the system creates posts that have the AVPair as the key and contain document information, and contact information of peers who have information about this document in its content. Attribute-value post (AVPost), is the post that is distributed in the global directory with the key specified by a hash of AVPair, $key = hash(attribute = value)$. Each AVPost contains information about a document corresponding to the AVPair, URL of the document, peer id who has information the document, peer id of the peer who insert the post (poster id), and email of the user who create this post. This type of index is aimed to support use case AD.

The second type of additional index in the design is aimed to support the use case DA. This type of index maps from document id to attribute values created for the particular document from all users in the system. The document id to attribute values index is formed by document id posts (DocIdPost) distributed from all peers over the network. DocIdPost is a post that is have the key specified the hash of the document id (docId), $key = hash(docId)$, and is created along with the AVPost. DocIdPost contains information about attribute, attribute value, poster id, and email address of the user who responsible for this AVPair.

To support global index extension in this design, we add a new database table called "P2P_ATTRIBUTES" in the MINERVA database schema. This table is created in order to store attribute values for documents created by the user in the local peer. Each entry in this table contains information about attribute, attribute value, URL of the document, document id of the document, and peer id of the peer who have this document in its document database. Both AVPost and DocIDPost are created for each entry in table P2P_ATTRIBUTES. On receiving a new attribute value from the user, the application automatically stores received attribute value along with other related information in its local database, creates AVPost, creates DocIdPost, and publishes both posts to the global directory. Figure 4.1 presents example of post's creation.

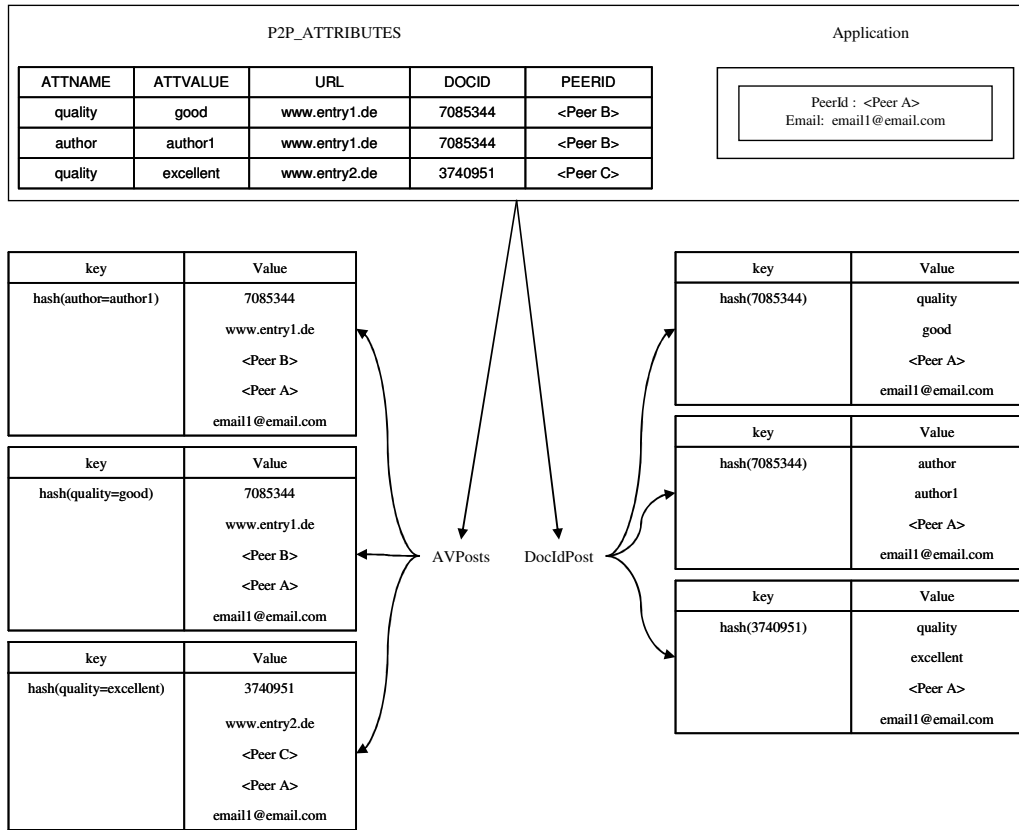


Figure 4.1: Example of posts created in PDI for peer <Peer A>

In the query process, when the query is submitted to the application, the application first splits the multi-term query $Q : \{t_1, \dots, t_m, a_1 = v_1, \dots, a_n = v_n\}$ into single term queries, $q_i : \{a_i = v_i\}$ or $q_j : \{t_j\}$. For each single term query $q_j : \{t_j\}$, the peer list is retrieved. Each attribute value query $q_i : \{a_i = v_i\}$ is processed as follow. The query's key is computed from a hash of AVPair, $key_i = hash(a_i = v_i)$. The application calls the function $lookup(key_i)$ in the DHT for the responsible peer. Lists of AVPost that related to each query key are retrieved from the peer and are merged into the single AVPost list. Each post in the list contains document id, URL, and peer id of the peer who have this document information. User can reach the document directly from URLs

retrieved with the post. If executing the query with only attribute values $Q: \{a_1 = v_1, \dots, a_n = v_n\}$, or executing the query that contains the combination of terms and attribute values $Q: \{t_1, \dots, t_m, a_1 = v_1, \dots, a_n = v_n\}$, the retrieved Peer list is matched with peer id containing in the retrieved AVPost list. The query is sent to be executed locally in the peers' local database with document ids as a filter. The final result is merged as same as in the current implementation of the MINERVA system.

If user want to retrieve the other attribute values related to the particular document, as describe in the use case DA, the operation is done by computing the query key by hashing a document id, $key = hash(docId)$, and call function $lookup(key)$ to find the responsible peer. List of DocIdPost is retrieved from the responsible peer.

In an example in figure 4.2, user wants to search for documents that contain attribute value “quality=good”. The peer looking up in the global index with the key computed from hash of AVPair. AVPost related is retrieved form the global directory. After these two steps, we have a result that contains document id, URL, peer id of a peer who has information about the document in its local database, poster id and email of the user who create this attribute value. User can retrieves documents directly from the URL or get all attribute values corresponding to the document as represented in figure 4.3.

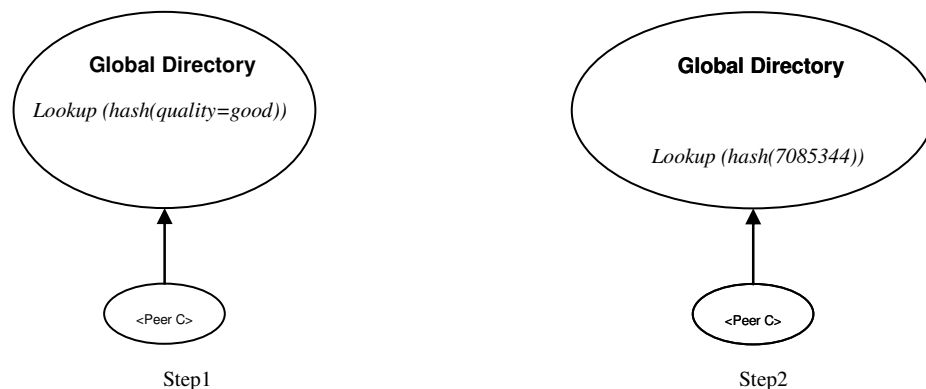


Figure 4.2: Example query process for query “quality=good”

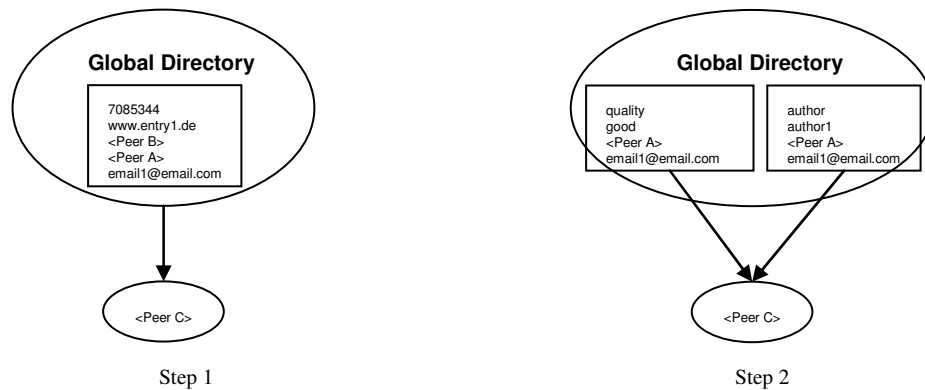


Figure 4.3: Example query process when user wants to retrieve more attribute values for document.

4.3 Document List Approach (DL)

This design is aimed to resolve a drawback of the PDI approach. In PDI approach, each time user insert an attribute value for the document, information about peer id that has information about this document in its local database has to be input manually by user or automatically by the application. Once the peer id that is specified by the user dose not exist or the other peers are also has this document stored in their local database, incorrect or incomplete query result might reported to the user.

In order to solve this problem, the global directory is formed my four type of index, index that maps term to peer list (standard MINERVA), index that maps AVPair to document ids, index that maps from special common key to document id list, and index that maps from document id to AVPairs. Global index that maps AVPair to the document ids is formed by attribute-value posts (AVPost) crated by all peer in the system. An AVPost is the post created from each attribute of the document stored in each peer and is distributed in the global directory with the key specified by a hash of AVPair as same as in PDI, $key_i = hash(a_i = v_i)$. Unlike PDI, AVPost in DL contain only a document ID corresponding to the AVPair in its content.

Similar in PDI, global directory contains index that maps from document id to AVPairs, to support the use case DA. This index is formed by document id posts (DocIdPost) created for each AVPair crated in all peers in the system. DocIdPost has a key specified by a hash of document id, $key = hash(docId)$ and contains attribute and attribute value corresponding to the document id in its content. Each DocIdPost is created for each AVPair stored in the local peer.

The last type of index in the global directory in this design is the index that maps from a special common key to document id list stored in each peer in the system. The index is formed by document id list posts (DocListPost) created from all peers in the system. DocListPost is a post that has a key specified by common special key that is agreed by all peers in the system and contains local peer id and a list of all document ids stored in its local database. This index allow user to get list of document stored on each peer by performing lookup operation with the common key.

Database table P2P_ATTRIBUTES is added to the MINERVA database schema to store document attribute values created by the user in the local peer. But unlike in PDI, that database table P2P_ATTRIBUTES is consisted of only three columns to store attributes, attribute values, and document ids. Each time user creates attribute for document; the application stores received information in the P2P_ATTRIBUTES and distribute posts related to the received information as same as in PDI. Figure 4.4 represents posts that are created in the second design.

Upon receiving query string from the user, the query is split into many single term queries similarly to PDI. The Query processes for attribute value query $q_i : \{a_i = v_i\}$ can be organized into 7 steps. Step 1, a hash of AVPair is used as a query key, $key_i = hash(a_i = v_i)$, then the peer calls the function $lookup(key)$, provided by the DHT, to find the responsible peer. Steps 2, lists of AVPost corresponding to each query key are retrieved from the peer and are

merged into the single AVPost list. Each AVPost contains document id of the document corresponding to the AVPair. In multi-term query, lists of document id retrieved in this step is then merged (by matching) into a single document id list. Step 3, the peer performs a lookup operation again with a common key for DocIdListPost, example: $key = hash(DocIdListKey)$. The peer retrieves a list of document id stored in every peer's local database, step 4. Then the peer search through every document id lists retrieved to find peer ids that contain document id retrieved in step 2, step 5. This step results in a list of peer id that stored the documents' information on their local database. The peer id list is merged with a ranked peer list retrieved from a term query $q_j : \{t_j\}$ in the standard MINERVA processes. In step 6, the query is sent to top-k peers in a merged peer list to be executed locally with document id retrieved in step 2 as a filter. The result is merged the same way as in standard MINERVA. The processes to retrieve all attribute values for a particular document is done in the same manner as in PDI.

Example of the query process is represented in figure 4.5. First, the peer <PeerC> wants to execute the query for documents that related to the attribute value "quality=good", the peer performs lookup operation in the global directory for AVPosts related. AVPosts those their key calculated from AVPair "quality=good" are retrieved from the responsible peer. At this step we know that a document with document id "7085344" is related to the given AVPair. Then we retrieve lists of document id stored in all peers in the system and search through all document id lists to find out which peer contains information of the document in its local database. In the example, we now know that the information of the document is stored in peer <PeerB> and <PeerC> (local peer). With the multi-term query, lists of peer id and peer lists from the term queries are merged into a single peer id list in this step. Then the query is sent to a peer <PeerB> and <PeerC> itself to be executed locally for information of the document with document id "7085344".

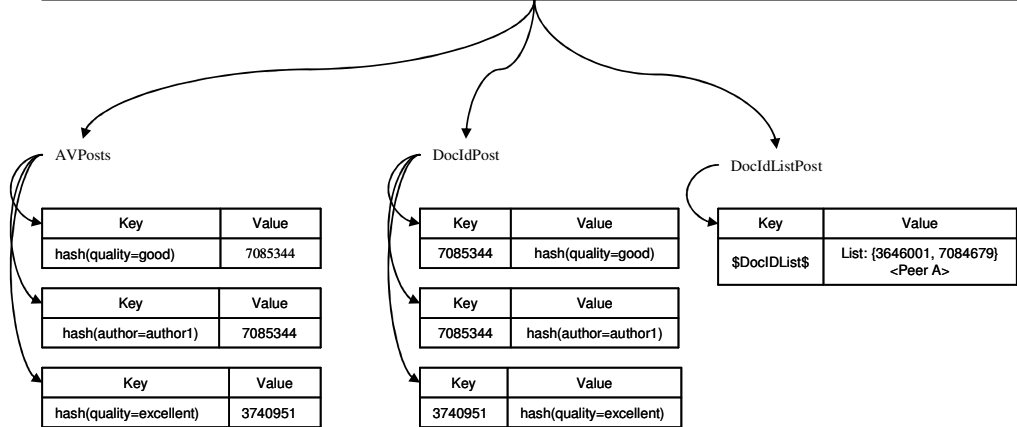
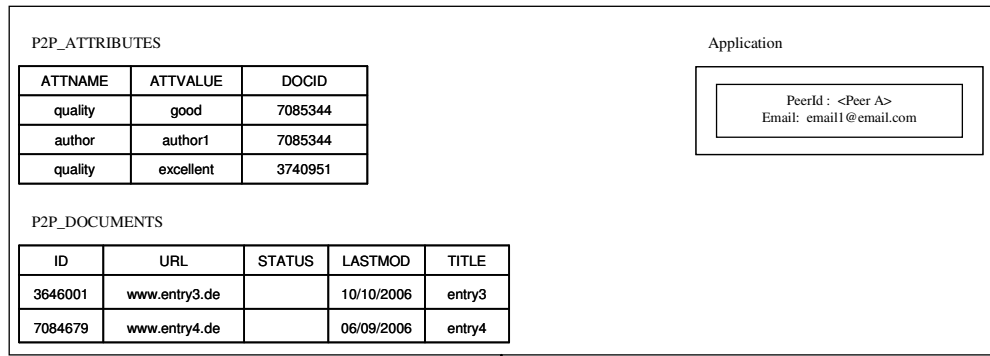


Figure 4.4: Example of posts created in DL for peer <Peer A>

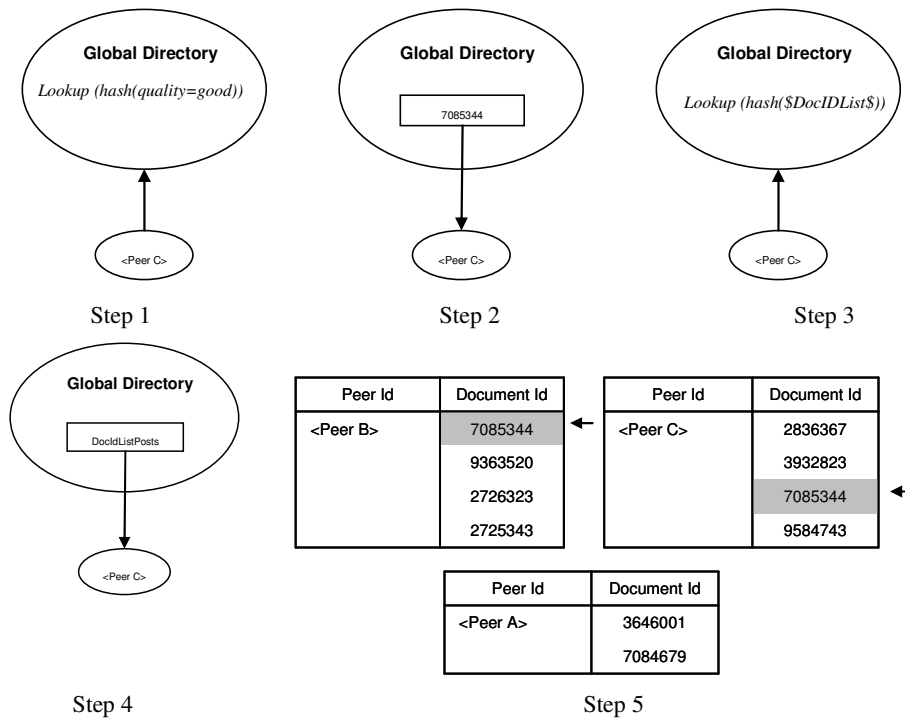


Figure 4.5: Query step 4-5 for DL approach

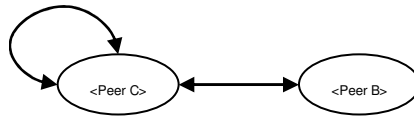


Figure 4.6: Query step 6 for DL

4.4 Full Document Index Approach (FDI)

Generally, this design is very similar to PDI, but it is designed to resolve the drawback of PDI approach that was mentioned in the last section. As same as in DPI, global directory consist of three types of index, index that maps from the AVPair to the document information, index that maps from the document id to AVPairs, and index that maps from term to peer list (standard MINERVA). The index that maps from AVPair to the document information is formed by AVPosts distributed from all peer in the system. An AVPost is the post that has the key specified by a hash of AVPair. But unlike in PDI, the AVPost in this design contains only related document id and URL of the document in its content.

The second index is the index that map from the document id. This global index if formed by document id post (DocIdPost) distributed form all peers in the network. There is two types of DocIdPost introduced in this design. Both types of DocIdPost share the same key which is a hash of document id, $key = hash(docId)$, but contain different object in its post content. The first type (DocIdPost type1) contains attribute and attribute value related to the document id in its content, this type of DocIdPost is crated for each AVPair stored in the local peer. The second type (DocIDPost type 2) contains a peer id of the local peer (poster id) in its content. DocIdPost type 2 is created one for each entry in the database table P2P_DOCUMENTS in the local peer. This type 2 post provide full index to all document information in

the system. Figure 4.7 presents posts create form information stored in an example peer.

The database table P2P_ATTRIBUTES is added to store attribute of the document inserted by the user on the local peer. P2P_ATTRIBUTES in this design have four columns; each entry consists of attribute, attribute value, URL, and document id. AVPost and DocIdPost typ1 is created for each entry in this table.

In the query execution of this design, after the multi-term query $Q: \{t_1, \dots, t_m, a_1 = v_1, \dots, a_n = v_n\}$ is split into many single-term queries $q_i: \{a_i = v_i\}$ and $q_j: \{t_j\}$ as same as done in PDI and the DL, the term query $q_j: \{t_j\}$ is processed as normally done in the standard Minerva to get a peer list. The attribute value query $q_i: \{a_i = v_i\}$ is processed as follow. An application performs a lookup operation in the global directory with the key specified by a hash of AVPair, $key_i = hash(a_i = v_i)$. AVPosts corresponding to the specified AVPairs are retrieved from the responsible peer. After finish this step we have a list of document id corresponding to the given AVPairs. In the query that contains only AVPair, documents can now be reached directly from the URL containing in post contents. Then the application performs a lookup operation in the global directory again with all document ids from the retrieved list in the last step to retrieve type 2 PeerIdPosts. This step results in a list of peer id that is related to the given document id. The list of related peer id from the last step is used in peer selection process by merging with the lists of peer id or peer lists retrieved from the other queries to get a single list of peer id. Query is then sent to the top-k peers in the peer id list to be executed locally for the query result. Query result is then merged in the same manner as the standard MINERVA. The process to retrieves all attributes for a document are done in the same manner as in PDI.

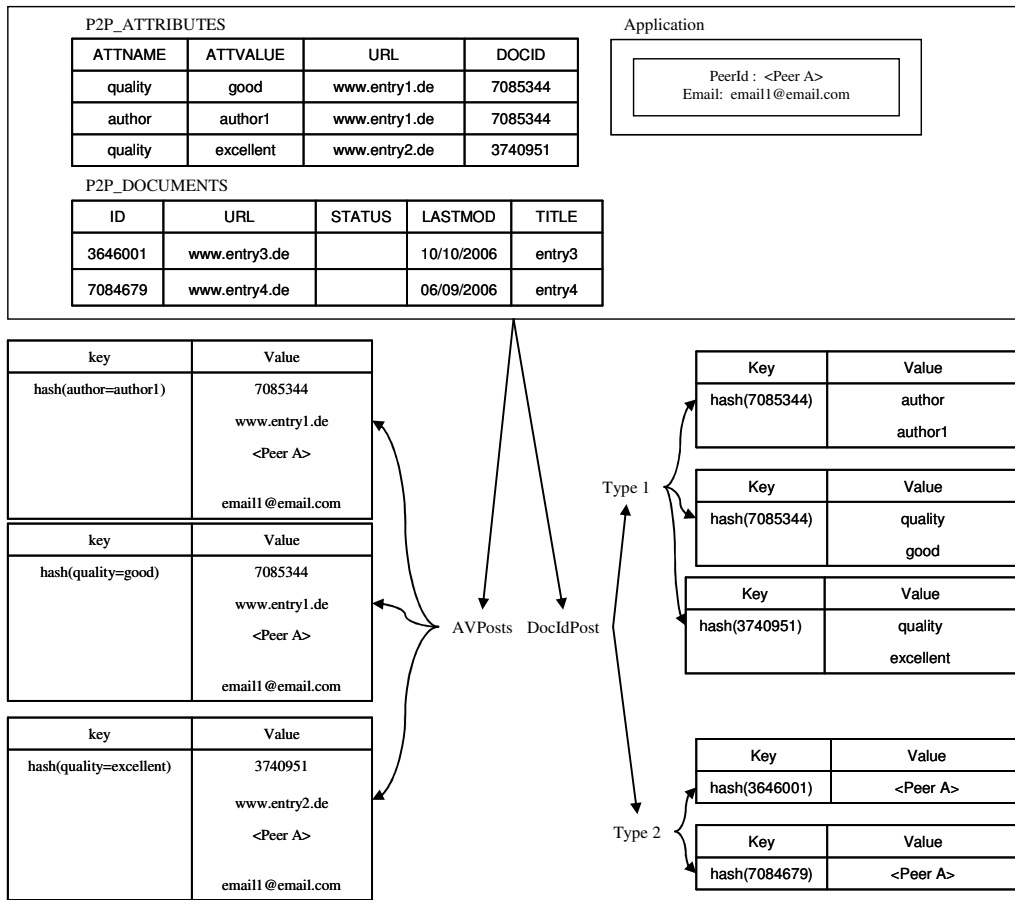


Figure 4.7: Example of posts created in FDI for peer <Peer A>

Figure 4.8 represents an example query with keyword “quality=good”. In step 1, an application performs operation $lookup(key)$ where $key = hash(quality = good)$. In the second step, AVPosts related to “quality=good” is retrieved from the global directory. Each post contains the document id corresponding to “quality=good”. After finish the second step, we now knows that the document with document id “7085344” is related and can be reached by the URL extracted from the post. In step 3, the application performs lookup operation in the global directory again to retrieve DocIDPosts type 2, to find out which peer has this document information, $lookup(key)$ where $key = hash(7085344)$. In step 4, DocIDPosts corresponding to the given document id are retrieved from the responsible peer, only information about

peer id in DocIdPost type 2 is used. In the example, we now know that peer <PeerB> and <PeerC> (local peer) have document with document id “7085344” in its document database. In the multi-term query, after each attribute value query finished this step, the result is lists of peer id related to each query. All peer id lists are then merged and matched with the ranked peer list retrieved from term queries. Then in the last step, query is sent to the peer in the list from the last step to be executed locally. Query results are received and then merged into a single result list.

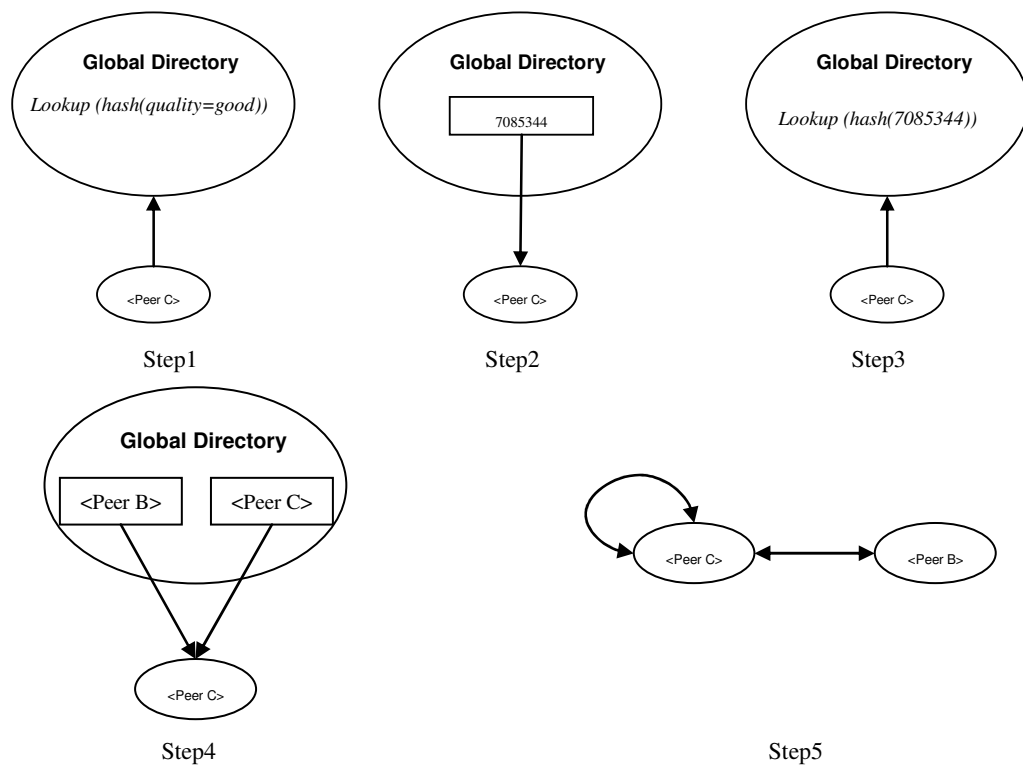


Figure 4.8: Query steps for FDI

4.5 Discussion and Decision

The PDI approach presents one solution to support queries with attribute and value pairs. Apart from the index that map term to peer list created in the standard MINERVA, an application also creates attribute and

value post and document id post for each attribute entry stored in the database table P2P_ATTRIBUTES in its local database to form an index that maps AVPair to the document information and index that maps the document id to AVPairs. In PDI, the total number of posts created in each peer is the summation of the number of terms and two times the number of attribute values stored in the local database. An application performs lookup operations only once for each AVPair in the query and only once when a user wants to retrieve attributes of the specific document. But there is one drawback of this design. AVPost that is created in this design, the information about the peer id of the peer who has information of the document is given manually by the user. This peer id is used as a filter in a peer selection step when making a query with a combination of terms and attribute values. For example, peer P1 and P2 both contain statistical information about document D1 and there is an attribute value (AVPair) AV1 that is related to document D1. And there is only one AVPost with the key AV1 and containing only peer id P1 in its content is created by some user in the system. When the user executes a query with AV1 and some term related to the document, peer P2 will be filtered out in the peer selection process.

To resolve the problem of PDI, in DL, we let each peer create a list that advertises all document ids stored in the local peer. AVPost contains only the document id of the document related. And then we search through the list for the peer who stored this document information. With this approach, there is no such a problem that some information is lost, that happens in the PDI. A drawback of the DL is related to the list of document id advertised by every peer. According to the design, every peer shares the same identifier to post this document id list. This approach creates a hot spot in the system, every query with the attribute-value pair needs to get a list of document id from the global directory, resulting in heavy traffic in the peer who is responsible for the common key for the document id list posts and the long list of document id also reduces searching performance.

FDI is aim to resolve the problem of PDI and DL. Instead of distributes a list of all document ids, FDI provides full index to all document ids stored in all peers in the global directory. Peer id of the peer that contains the document id specified by the user can be retrieved directly from the global directory. A drawback of this design arises from a number of post created. Posts are created in a summation of the number of term, number of document, and two times number of document attribute value stored in each peer's local database. This huge number of data to be posted in to the global directory may lead to inefficiency of searching in the global directory.

After many discussions, we have to become clear to our assumptions of the system that the number of attribute value is very small, compare to the number of document and term. The application is proposed to be used in a limited environment such as a research center. The DL approach leads to inefficiency according to long list search and also creates a hot spot in the network. The problem of PDI becomes less important as users normally satisfy with the partial query result. There is not a big different to get all peers who have the document information or only one peer who has the document information as long as it is the same document and with the assumption that the document information is stored permanently in the peer. FDI approach creates a lot more posts in the global directory than the PDI and according to the assumption it is not very value to waste the space as in the FDI. With all advantages, disadvantages, and all assumptions we have made, we conclude to implement the PDI approach..

Chapter 5

Extending MINERVA

In this chapter, we present parts of the MINERVA system that need to be extended in order to provide features for a document recommendation system based on attribute value search.

5.1 Global Index Extension

In the standard MINERVA system, the global directory is formed by indexes that map from term to peer list. In our PDI approach, we introduced two types of global index that need to be added in order to provide ability to make a query with attributes of the document. The global index that maps AVPair to document information is formed by Attribute-value posts (AVPost) distributed for each AVPair by all peers in the system. The key of an AVPost is calculate from the hash of the AVPair, a string that generated by concatenate attribute and attribute value with “=” sign in between, e.g. quality=good. Each AVPost contains information about a document id corresponding to this AVPair, URL of the document, peer id that containing this document in its local database, peer id of the local peer (poster id), and email of the user who creates this post. This type of post is used when the user want to make a query using attribute values as a keyword, as described in the use case AD. Another type of index that is mentioned in PDI is an index that map from document id

to AVPairs. This type of index is formed by document id post (DocIdPost) distributed from each attribute entry stored each peer in the system. DocIdPost is a post that is keyed by the hash of the document id. Each DocIdPost contains information about attribute, attribute value, peer id of the local peer (poster id), and email address of the user on the local peer in its content. This type of index is used when the user wants to retrieve attribute values related to a specific document.

The final global directory of the MINERVA System after all extension consists of three types of global index as mentioned above each index responsible for each type of query. The global index that maps terms to peer list is used for the term queries, the index that maps AVPairs to document information is used for attribute queries, and the index that maps from document ids to AVPairs is used to retrieve attribute values.

Dataflow diagram presented in figure 5.2 shows processes to generate AVPosts and DocIdPosts and distribute them in the global directory. Each attribute entry that is stored in the local database contains information about attribute, attribute value, URL, document id and peer id. This information is retrieved in process 1 and is used to create DocIdPosts and AVPosts in process 2 and 3 respectively. In process 2, DocIdPost content is created from attribute and attribute value received from process 1, email address and local peer id (as poster id) from the application. Users must specify the email address during the joining process and the local peer id is generated directly by Pastry from the user's email address. DocIdPosts created for each AVPair in the local peer are then distributed in the global directory. In process 3, AVPost content is created from URL, document id and peer id from process 1, email address and poster id is received from application as same as in process 2. AVPost is then distributed in the global index with identifier created from attribute and attribute value as mentioned before.

5.2 Database Extension

The document database schema of the standard MINERVA system consists of four database tables, P2P_DOCUMENTS, P2P_FEATURES, ARCHIVEFILES, and P2P_STATISTIC. These tables provide statistical information used to calculate IR style score and to generate indexes that map from term to the local peer to be distributed in the global index. In our design, document recommendations are specified as a type of attribute of the document, and attributed created by a user on a local peer need to be stored on the local peer. In order to support global index generation that is mentioned in the last section, a database table P2P_ATTRIBUTES is added to the MINERVA's database schema to store information about all attributes created on the local peer. The database table consists of five columns; each entry in the table consists of information about attribute, attribute value, URL of the document, document id, and peer id of the peer who has information about the document in its local database. Each entry stored in the database table P2P_ATTRIBUTES is used to create one AVPost and one DocIdPost. Figure 5.1 represents an extended database schema for MINERVA. Figure 5.2 represent the post generation dataflow diagram.

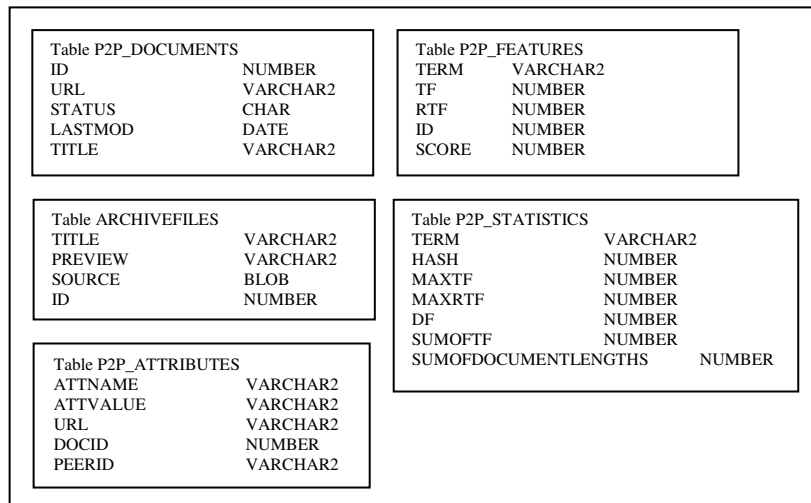


Figure 5.1: Extended database schema in MINERVA

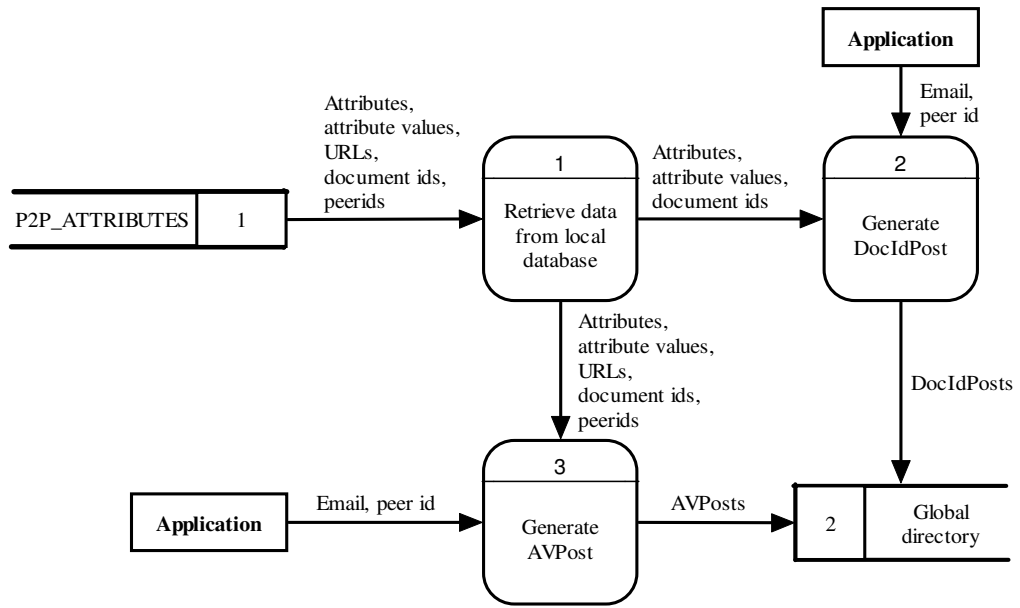


Figure 5.2: Dataflow diagram of an extension part to generate AVPosts and DocIdPosts

5.3 Query Processor Extension

Another part of MINERVA that we need to extend is the query processing part. In the standard MINERVA, users can only submit term queries in the form of $Q: \{t_1, \dots, t_m\}$. Unlike standard MINERVA, users' queries in an extended system can consist of terms and attribute value pairs. Those queries in the extended system can be categorized into three types: queries that contain only terms $Q: \{t_1, \dots, t_m\}$, query that contains only attribute value pairs $Q: \{a_1 = v_1, \dots, a_n = v_n\}$ and, queries that contain both terms and attribute value pairs $Q: \{t_1, \dots, t_m, a_1 = v_1, \dots, a_n = v_n\}$. Each type of user query needs to be processed in the different way and additional processes to realized and split term query and attribute query need to be added to the system. The keyword in the query is realized as an attribute if the character “=” is detected between

attribute and attribute value. For example, user query “lsi quality=good” can be split into two single term query, “lsi” and “quality=good”. The query with the keyword “lsi” is a term query and query with the keyword “quality=good” is the attribute query. This user query is then processed as the query that contains both terms and attributes. Many processes are added to handle query processing. The application performs lookup operation in the global directory to retrieve a peer list for each term query as same as in the standard MINERVA.

Figure 5.3 represents the dataflow diagram of query processing. According to the diagram, in the process 1, when the query is submitted to the application, query is not only split into multiple single term queries, but also categorized into two types of query, term query $q_j : \{t_j\}$, and attribute query $q_i : \{a_i = v_i\}$. If the user query is recognized as term query $Q : \{t_1, \dots, t_m\}$, single term queries $q_j : \{t_j\}$ are sent to process 2. Process 2 performs lookup operation in the global directory for a peerlist for each query. Peer lists are sent to process 4 to merge into a single peerlist ordered by IR-style score, based on statistical information calculated in term query processing (MINERVA [11]). Ranked peer list is received in process 5 and the query is routed to top-k peers to be executed locally. Peers return query results as set document information. Each entry in the result set contains title, URL, document id, peer id and last modification date of the document. This result set is then reported to the user. If the user wants to retrieve other attribute values related any document in the result set, document id is sent to process 6 to perform lookup operation in the global directory, keyed by document id. The global directory returns DocIdPosts that identified by the document id, attribute and value pairs (AVPair) and email addresses are extracted from the content of posts and report to the user.

In process 1, if the user query is recognized as attribute query $Q : \{a_1 = v_1, \dots, a_n = v_n\}$, single term queries $q_i : \{a_i = v_i\}$ are sent to process 3

to perform lookup operation in the global directory based on AV pair to retrieve AVPosts related. All AVPosts retrieved is merged in to a single result set, URLs, document ids, and peer ids extracted from AVPosts retrieved are then reported to the user. If user wants to retrieve attribute values corresponding to any document in the result set, document id is sent to the process 6 to perform an operation as mentioned before.

If the user query is recognized as the query that is a combination of both terms and attributes $Q: \{t_1, \dots, t_m, a_1 = v_1, \dots, a_n = v_n\}$, single term queries $q_j: \{t_j\}$ are sent to process 2 to retrieve ranked peerlists, single attribute queries $q_i: \{a_i = v_i\}$ are sent process 3 to retrieve AVPosts. AVPosts retrieved in process 3 are then merged into a single list. A merged result list in form of document id and peer id pairs are sent to process 4 to merge with the ranked peer list retrieved from process 2 by matching peer ids. Matched peer id list is then sent to process 5 to route the query to the top-k peers to be executed locally with document ids retrieved from process 3 as a filter for result sets.

5.4 Attribute Input System

In order to support use case AC, the other part of MINERVA that needs to extend is the attribute input system. This part of the system provides feature for user to insert and distribute attribute values to documents. The dataflow diagram of the input system is represented in figure 5.4. To create an attribute of the document, user need to specify attribute, attribute value, URL of the document, document id, and peer id of the peer who has this document information in its local database. URL, document id, and peer id can be specified automatically by an application. According to the diagram in figure 5.4, information specified by user is sent to the process 1 to be stored as an entry in the table P2P_ATTRIBUTE on a local database and it is also sent to the process 2 to create AVPost and DocIdPost to update the global index. In

order to create AVPost and DocIdPost, email address of the user who creates an attribute of the document and peer id generated from the user email address are received from the application.

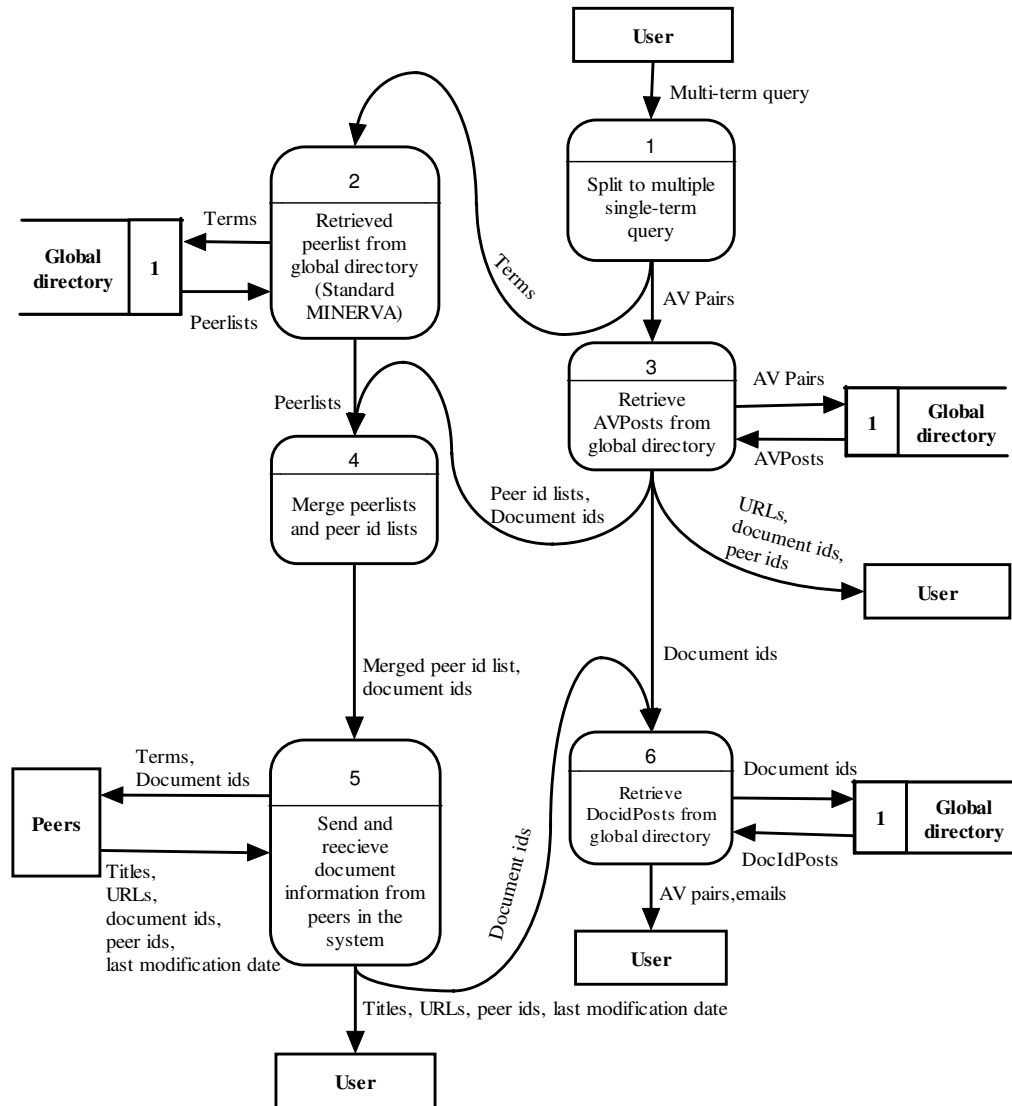


Figure 5.3: Dataflow diagram of query processing of extended MINERVA

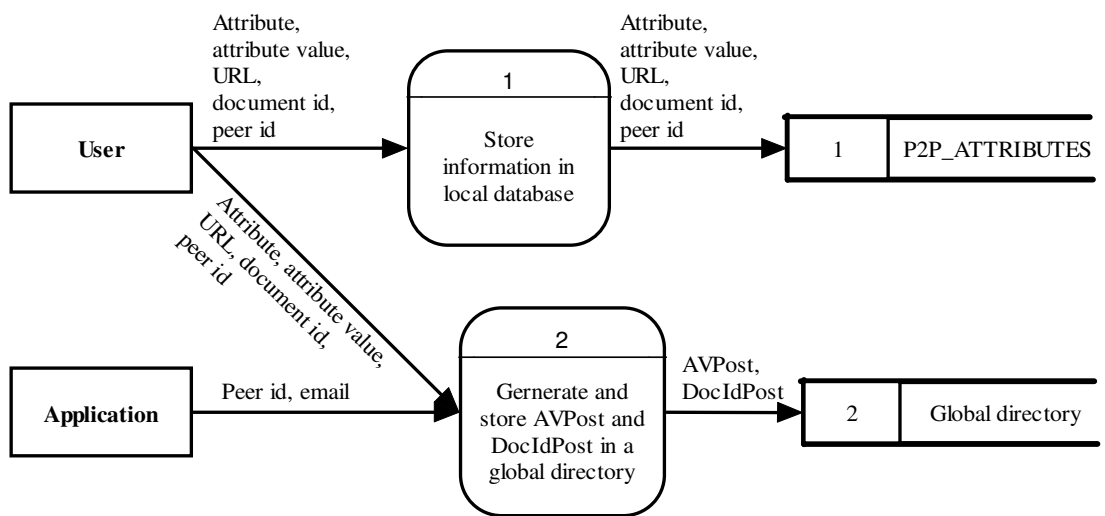


Figure 5.4: Dataflow diagram of recommendation input part

Chapter 6

Implementation

In this chapter, we present description of classes and methods that implemented in the MINERVA system. All classes and methods are implemented in Java 2 SE 1.5.0 and peers' document databases are maintained in Oracle 10g.

6.1 Class and Methods Involve in Pastry Connection

The standard MINERVA is implemented on top of Chord, one task of our work is to debug the MINERVA implemented in Pastry. Class `CreateJoin` is implemented to operate a Pastry ring creating or joining operation and database login operation, to form a peer in the MINERVA system. Two methods are implemented in the class:

- `CreateJoin.login` method receives information from the `LoginDialog` in the GUI and performs operations to form a peer in the Pastry ring. The method contains two parts to create Pastry ring or joining the Pastry ring and to login to the document database on the local peer.

- `CreateJoin.getNode` method returns Pastry node created in `login`.

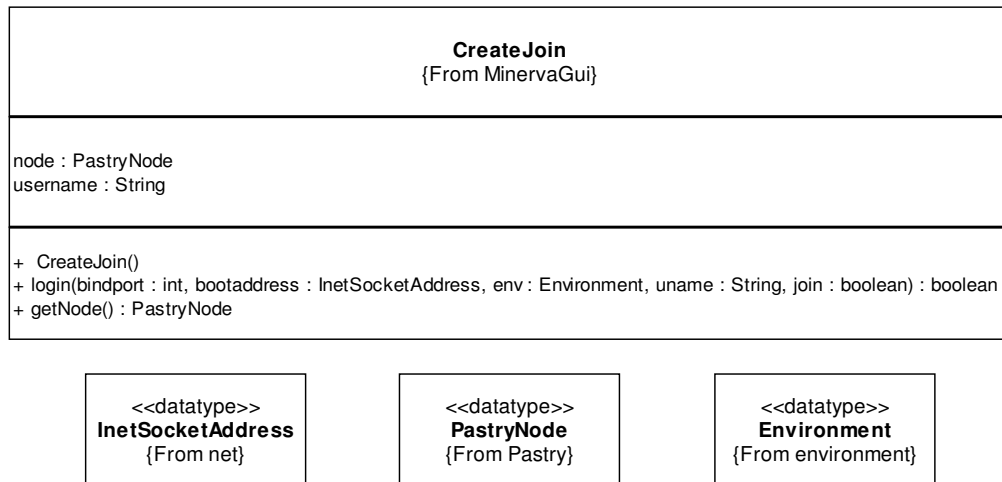


Figure 6.1: Dependency diagram of class `CreateJoin`

6.2 Class and Methods Involve in Global Index Generation

In order to handle three types of index presented in the selected design, we implement a class to handle these posts to form a global directory in PAST. Class `MinervaPastContent` is implemented on top of `GCPastContent`, this class is created for each identifier that this peer is responsible for in the global directory, is called automatically by PAST when post is distributed to this peer. Attribute `content` stores an array of post content with the same identifier. Three types of post are implemented to be stored in `content`.

- Class `PastPost`, this class is implemented in the standard MINERVA to maintain the peer list and statistical information related to term.

- Class `PastAVPost`, created for each attribute and value pair, indexes from attribute and value pair to URL, document id, peer id, poster peer id, and poster email as class attributes.
- Class `PastDocIdPost`, created for each attribute and value pair, indexes from document id to attribute, attribute value, poster peer id, and poster email as attributes.

Three objects of class above are wrapped into class `PastPostInterface` object and placed in the attribute `content` in class `MinervaPastContent`. When a new post arrives at the peer, PAST creates new instance of `MinervaPastContent` and call `MinervaPastContent.checkInsert` with the same identifier and its existing content as an input. The method first checks a type of content of class `MinervaPastContent`, if it is a content of `PastPost` or `PastDocIdPost`, class content is compared with the exist content to check if it is already existed. If the type of the content is the content of `PastAVPost`, the method first checks the delete flag in the content. Delete flag is set to true if user want to remove the same content as in the post from the global directory. If the delete flag is true, the method searches through an existing content for the similar content with class's content and removes it from the exist content. If the delete flag is set to false, the method searches and adds or updates class's content in the same manner as done for `PastPost`. Figure 6.2 represents processes in this method when the new post is arrive. Figure 6.3 represents UML class diagram related to all posts implemented.

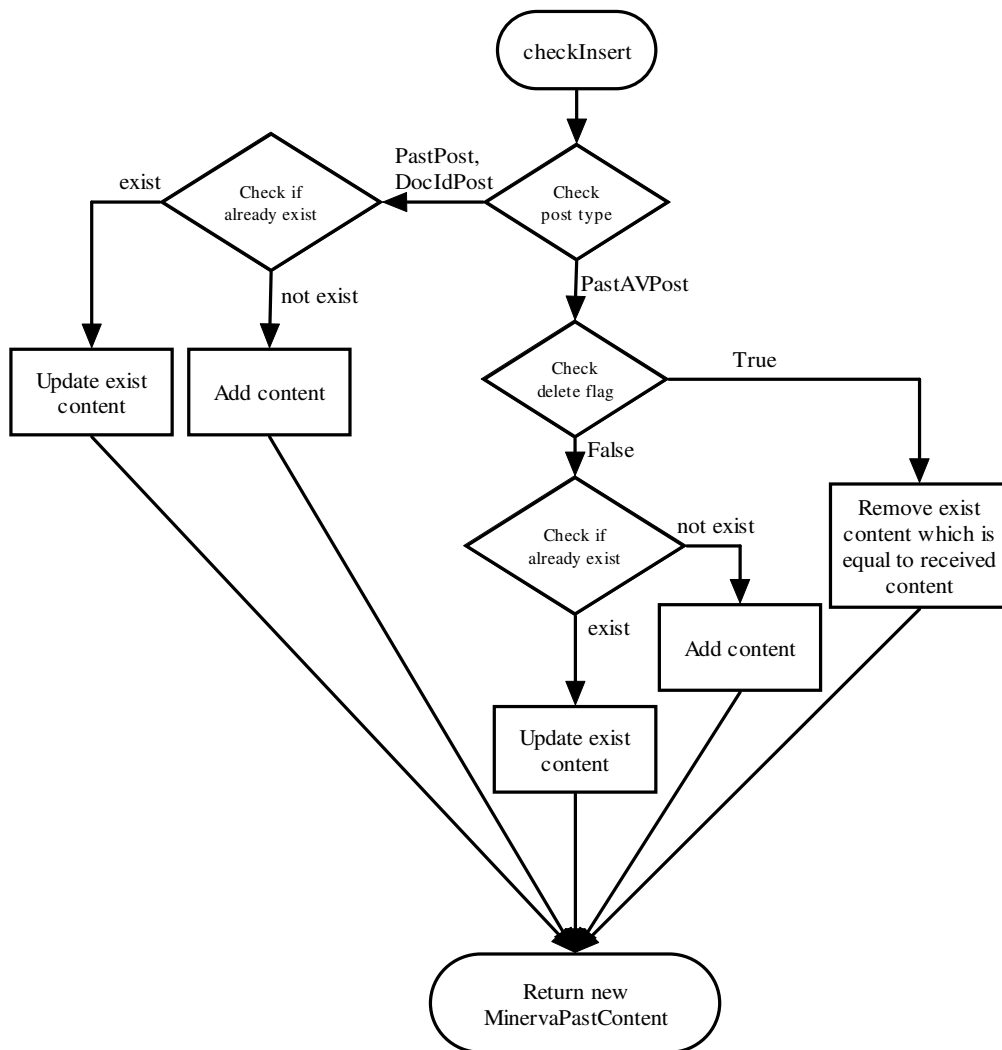


Figure 6.2: Flow chart of MinervaPastContent.checkInsert

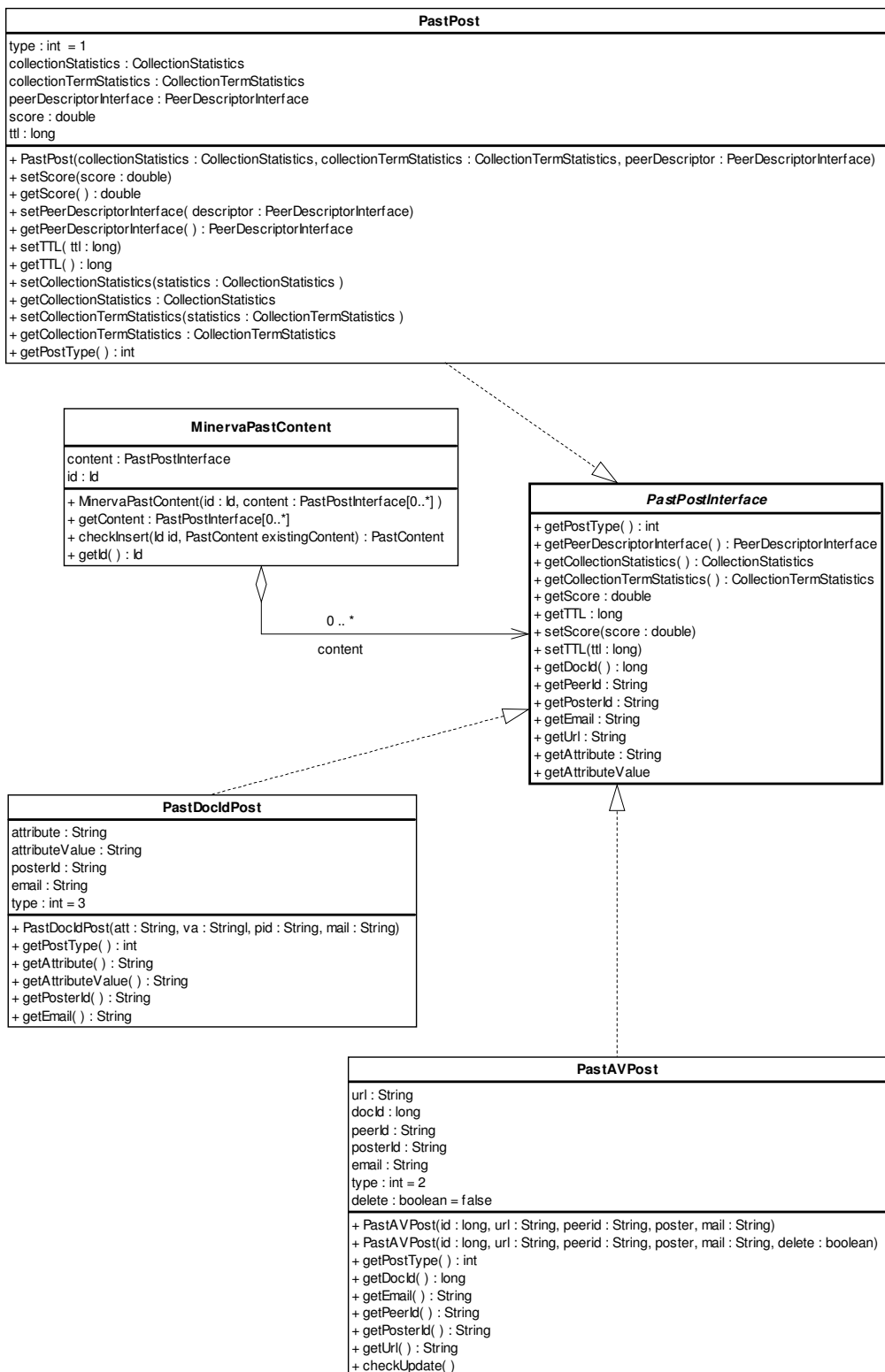


Figure 6.3: UML class diagram related on posts

6.3 Classes and Methods Involve in Query Processing

Apart from standard MINERVA system, in our design, user query can be terms, attributes or combination of both terms and attributes. Several methods are implemented mainly in class `Peer`, `PastGlobalQueryProcessor`, `EnhanceLocalQueryProcessor`, and `MinervaGui`. In the class `MinervaGui` many methods are implemented to prepare query to be executed later in the global query processor and local query processor. Important method implemented is:

- Method `MinervaGui.extractQueryTerm` is added to split and to realized term query and attribute query.

Several methods are added to `PastGlobalQueryProcessor` to provide features for attribute value based search. Some important methods are described below.

- Method `PastGlobalQueryProcessor.excute2` is called for attribute query. The method return a list of `PastAVPost` related to received query.
- Method `PastGlobalQueryProcessor.excute3` is called for a query with the combination of both terms and attributes.
- Method `PastGlobalQueryProcessor.mode3MergePeerList` merges peer list from term query and peer is list from attribute query into a single peer list.

- Method `PastGlobalQueryProcessor.getAVPost` retrieves and merges `PastAVPost` list for each keyword.
- Method `PastGlobalQueryProcessor.SendQueryMode3` generates query for term with document ids as a filter and sends the query respect to retrieved peer list.
- Method `PastGlobalQueryProcessor.RetrieveAV` is called when user wants to retrieve all attribute and value pairs for a specific document id.

In class `EnhanceLocalQueryProcessor`, one important method is added to execute the query with combination of terms and attributes

- Methods `EnhanceLocalQueryProcessor .excute3` executes the received terms query and a list of document id with the local database using document ids in the list as a filter.

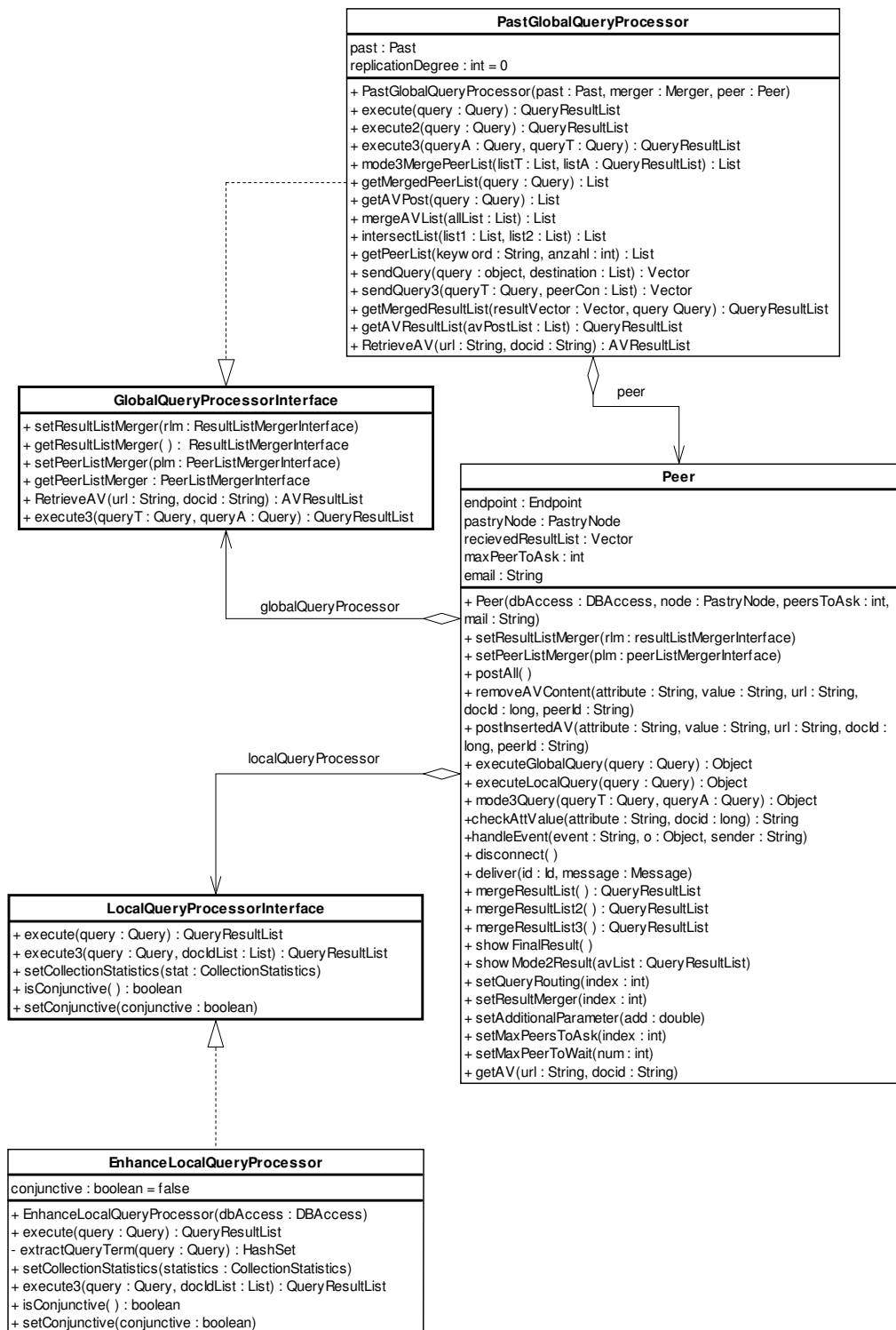


Figure 6.4: UML class diagram related in query processing

6.4 Methods Involve in Database Connection

Document database in MINERVA is extended by adding a database table `P2P_ATTRIBUTES` to store document attributes created on the local peer. The schema of extended database is represented in figure 5.1. Several methods are added to class `p2psearch.db.DBOracleAccess` to perform operations related to the attribute table.

- Method `DBOracleAccess.getAttributes` returns all entries stored in the database table `P2P_ATTRIBUTES` for the propose of global directory forming.
- Mehtod `DBOracleAccess.insertAttributes` generates new entry from the receiving parameters and adds or updates to the table `P2P_ATTRIBUTES`.
- Method `DBOracleAccess.computeCompleteQuery3` executes the query with input document ids as filter.
- Method `DBOracleAccess.setAttributeTable` is called during creation of peer to check if there is an existing database table `P2P_ATTRIBUTES` in the local database schema. If the table is not existing, new table is added to the local database.

6.5 GUI

In order to support an extension attribute searching, we implement a new GUI for the MINERVA system. All GUI classes are implemented in package `minervaGUI`. UML class diagrams of implemented GUI are presented in figure 6.12 and 6.13.

- Class `minervaGUI.MinervaGui` implements a main frame of the GUI
- Class `minervaGUI.NetworkInfo` implements a frame to display a network information such as local peer id and local database SID to the user.
- Class `minervaGUI.IncomingRequestFrame` implements a frame to display a list of incoming request for a local query execution from other peers in the system.
- Class `minervaGUI.Collectionstat` implements a frame to display local collection statistics. Figure 6.6 represents the main MINERVA frame, network information frame, collection statistics frame, and incoming request frame.
- Class `minervaGUI.LoginDialog` implements a dialog to get necessary information from a user to create pastry ring, joining pastry ring and login to a local database. Figure 6.7 represents the screenshot of login dialog.

- Class `minervaGUI.OptionDialog` implements a dialog that allow user to choose routing protocol, result merger, and number of peer to send query to. Figure 6.8 represents the option dialog.
- Class `minervaGUI.AVInputFrame1` implements frame that allow user to add attribute of the specific document manually. Figure 6.10 represent the attribute input frame for manual insertion.
- Class `minervaGUI.AVInputFrame2` implements a frame to receive attribute of the document from the user with preset document information. Figure 6.11 represent the attribute input frame with document preset value linking from the result frame.
- Class `minervaGUI.ResultFrame` implements a frame to display the query result.
- Class `minervaGUI.AVFrame` implements a frame to display document attributes retrieved for a specific document. Figure 6.9 represent result frame and attribute frame.

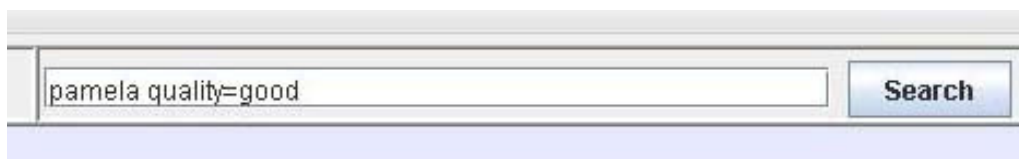


Figure 6.5: Example query in MINERVA GUI

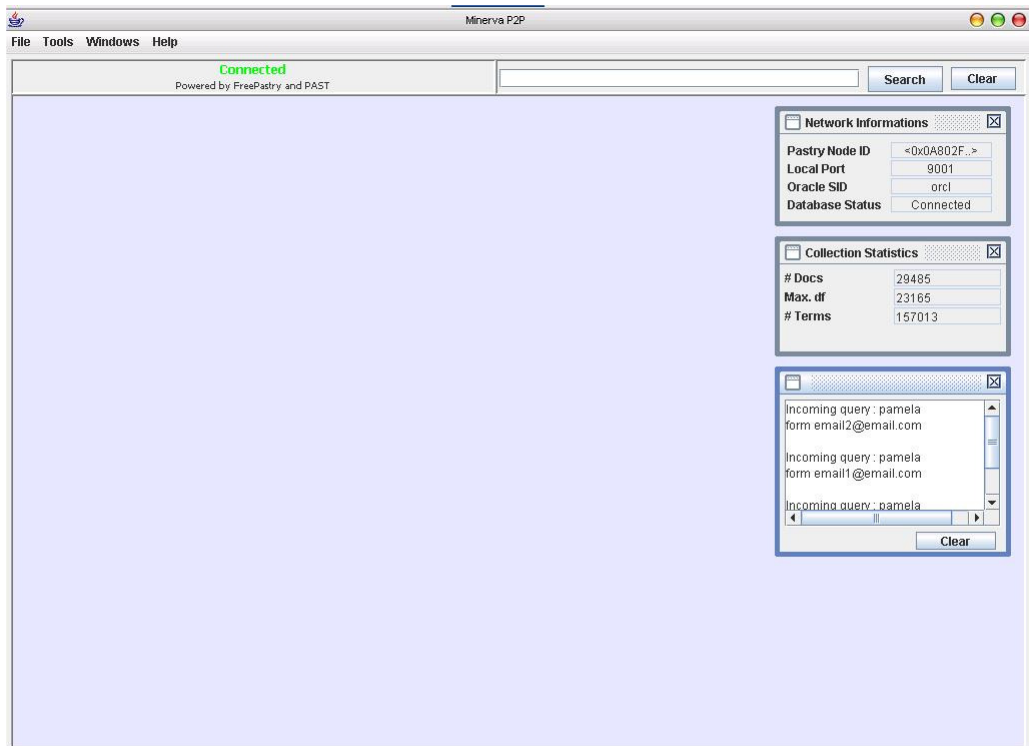


Figure 6.6: Main GUI of MINERVA and local peer's information windows

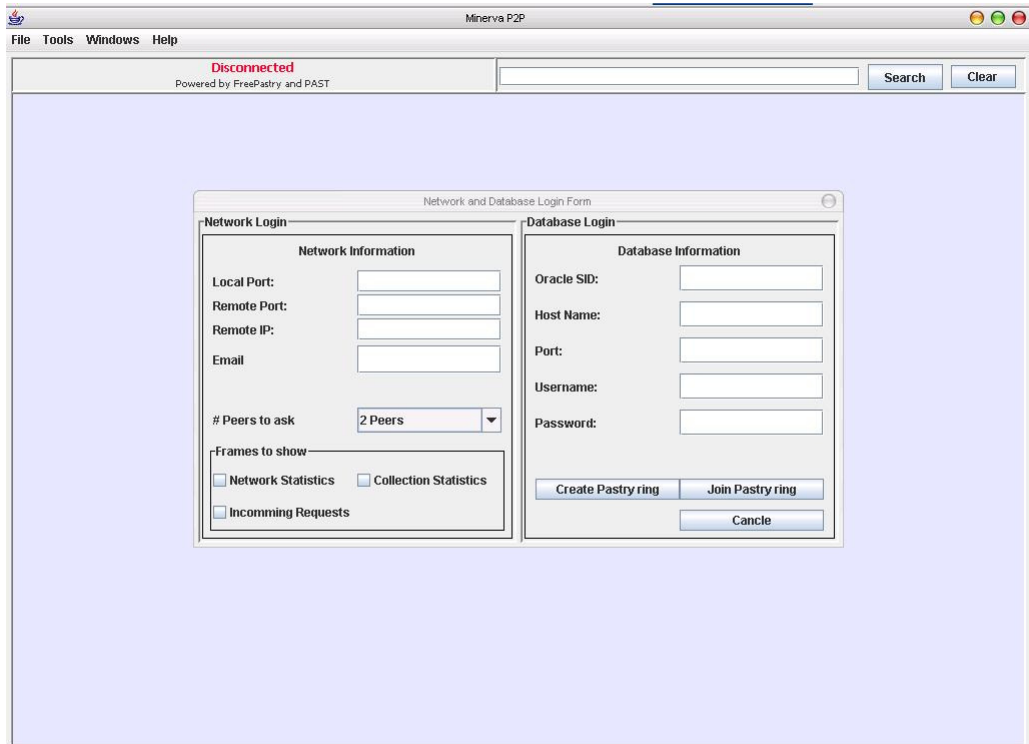


Figure 6.7: Login dialog

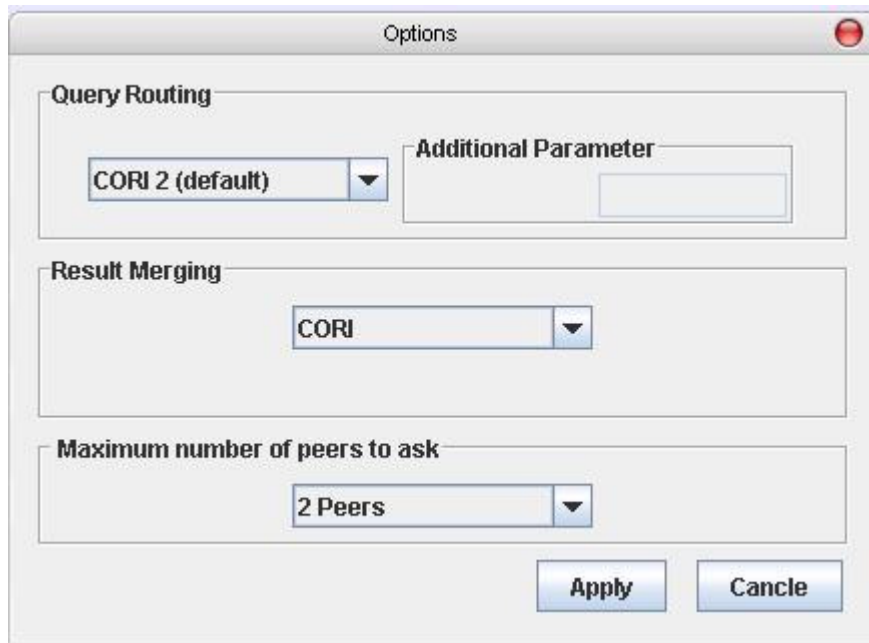


Figure 6.8: Option dialog

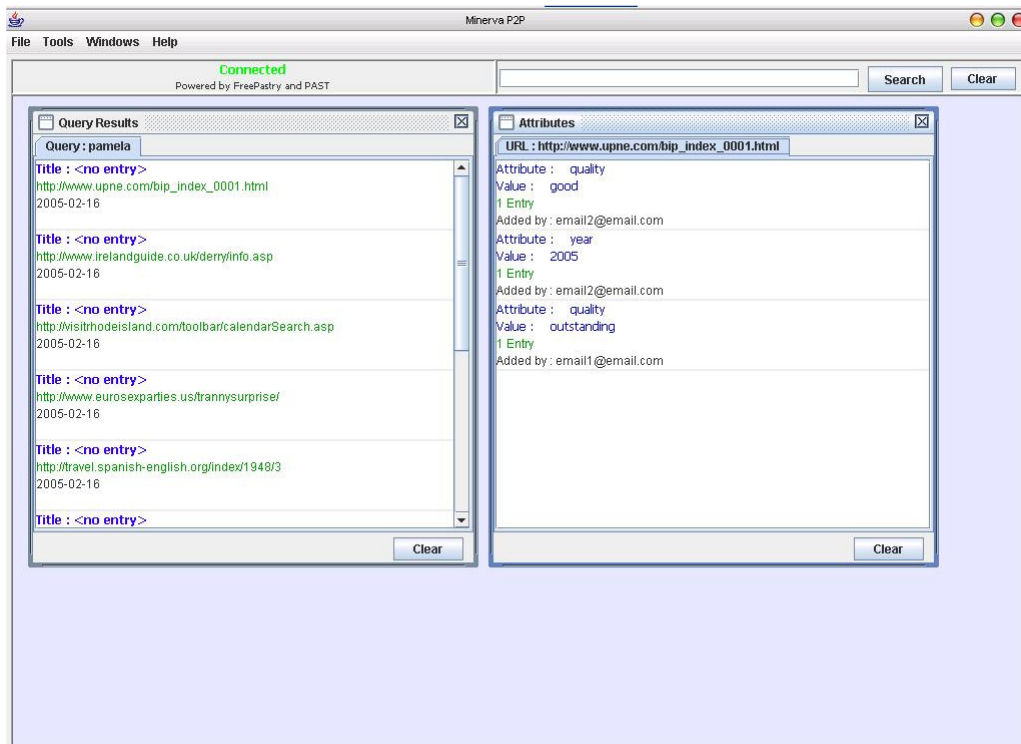


Figure 6.9: Query result reporting frame

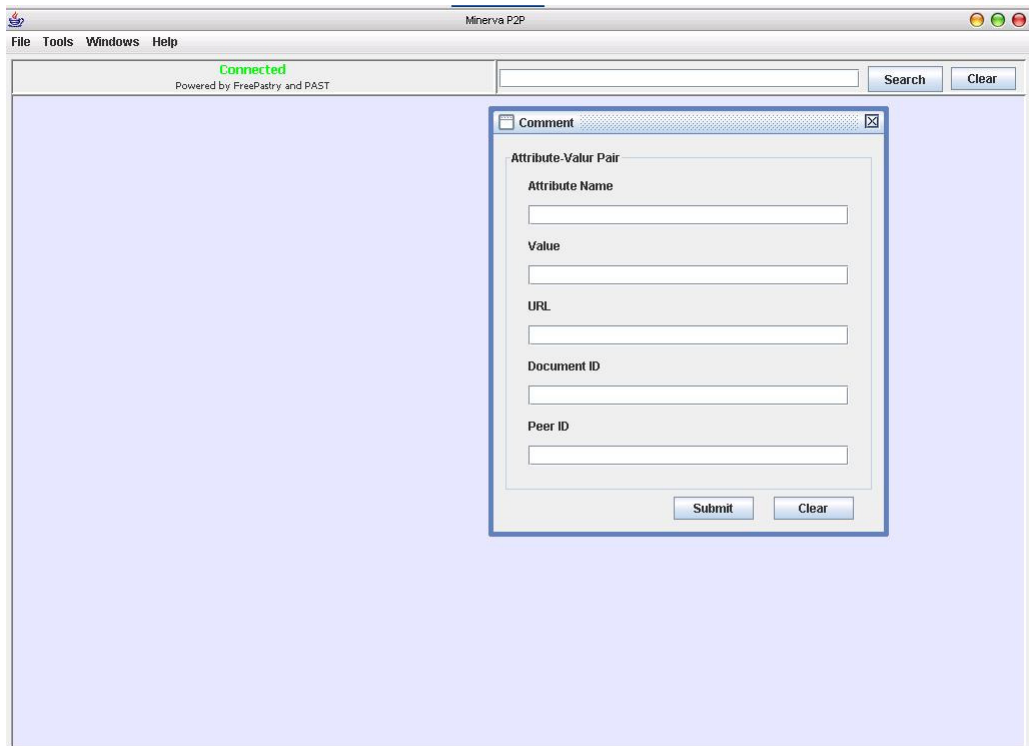


Figure 6.10: Attribute input frame (Manual)

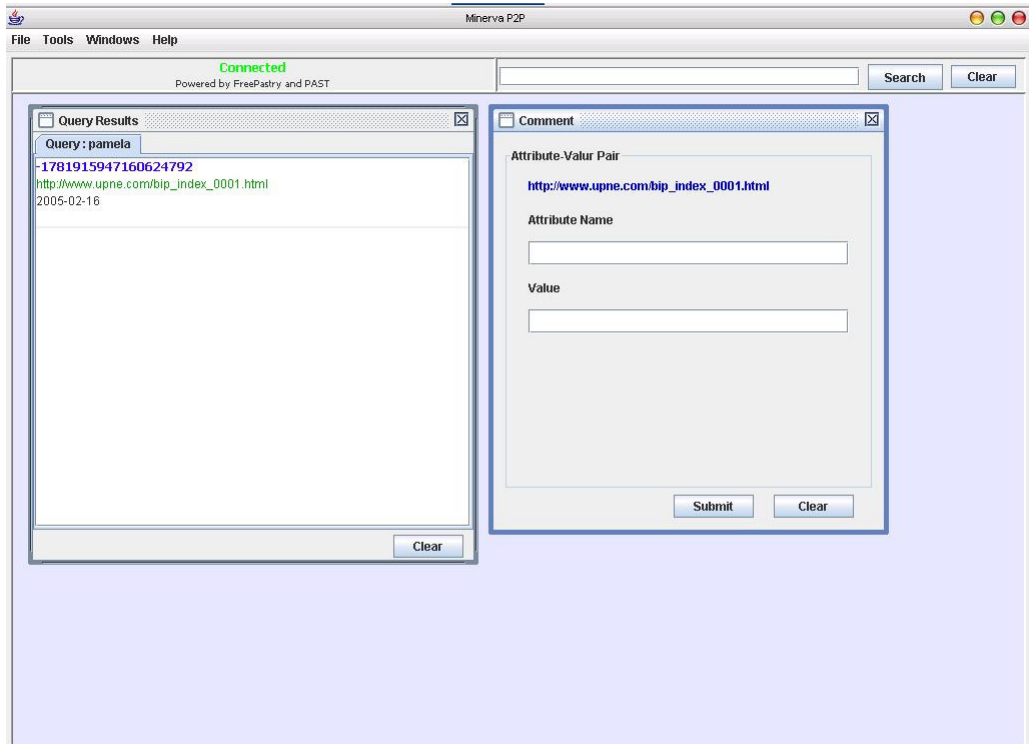


Figure 6.11: GUI, attribute input frame linking from query result frame

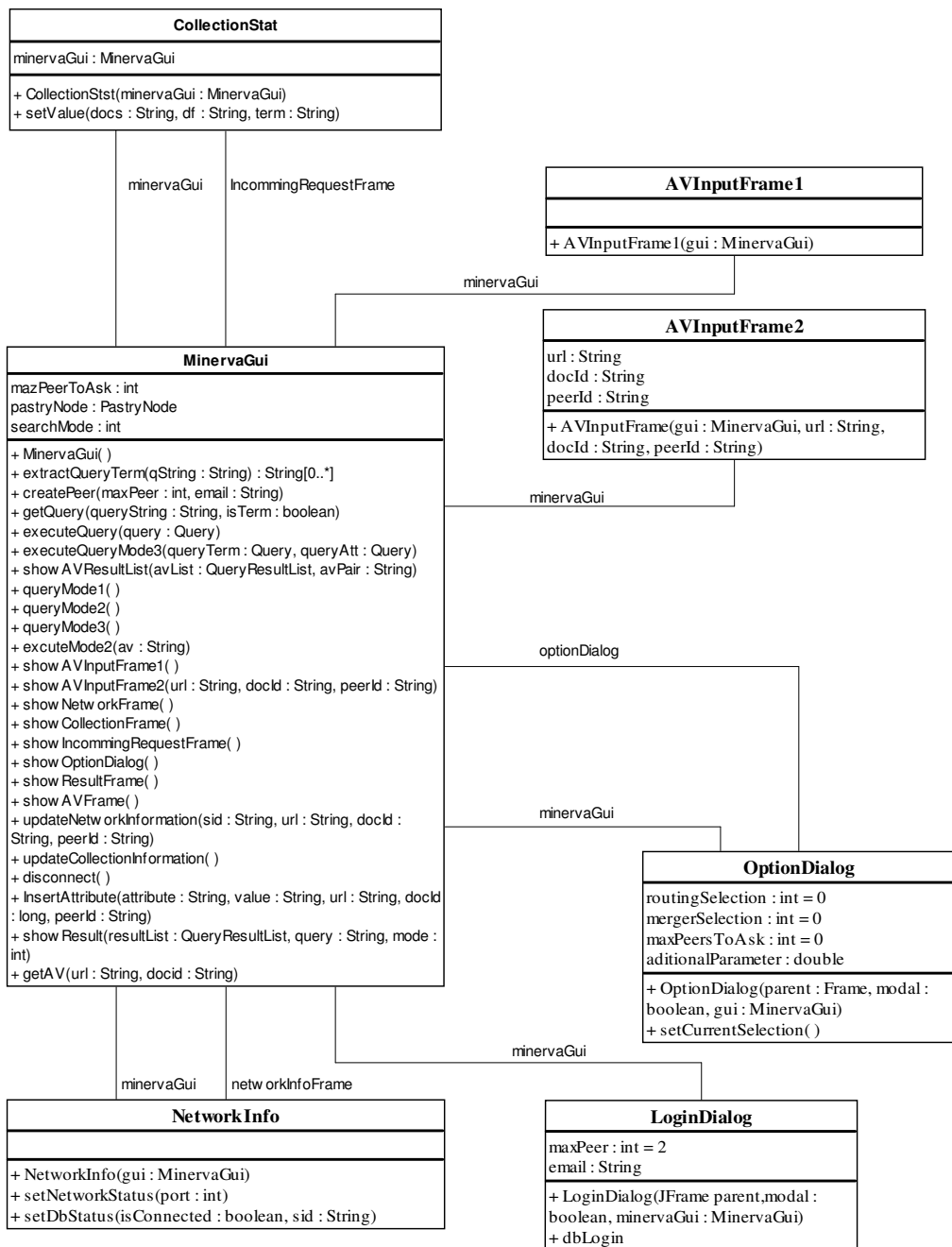


Figure 6.12: UML class diagram on GUI of MINERVA

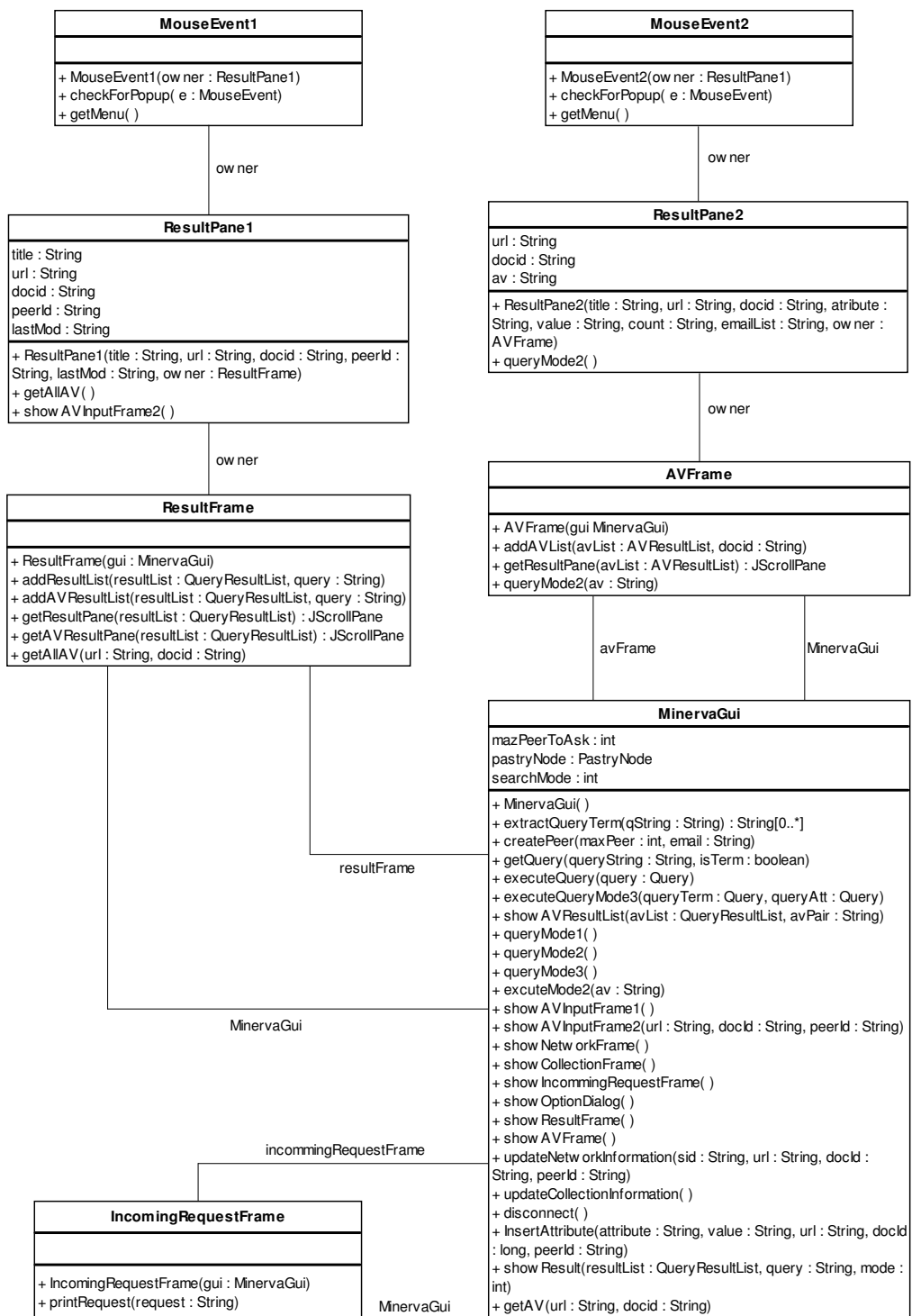


Figure 6.13: UML Class diagram of GUI involve in query input and output

6.6 Problems

During implementation we encountered some problems mostly related in migrating from Chord to Pastry. The first problem occurs at the joining process to the Pastry ring. An exception “Message dispatch” occurs when new node wants to join the Pastry ring that some nodes in the Pastry ring are already distributed some posts in the global directory. When this error occurs, some information stored in the global directory is lost. To resolve this problem, we tried to find out some delay to be inserted in each step of joining process. This solution did not help much the problem still occurs some times.

The other problem is occurring in the post process. When one peer in our test environment (3 peers running in the same machine) distribute posts to the global directory, an exception is start to occur at around the 3000 items, probably caused of too many messages have been post continuously into the global directory. Posts distributed after the first error has occurred some time can be stored in the global director without problem, but most of the time, posts are failed to store in the global directory.

Chapter 7

Conclusion and Future work

7.1 Conclusion

In this thesis we concentrated on extending the MINERVA system to support attribute value based search as well as related application such as bibliographic searching and creating a recommendation system. We review several existing approaches that related to our work and selected some useful approach to be used in our design. In our work, we also debug some part of the system that is migrated from Chord overlay network to Pastry overlay network

We propose a new global index to provide feature for attribute searching and attribute retrieving for the specific document. We implemented three types of index in our new global index. Indexes that map term to peer list from the standard MINERVA is remain the same to perform a query operation for term query. Indexes that map from AVPair to document information is implemented to use in the query that user want to get documents corresponding to the given document attributes. Indexes that map from document to AVPair is used to retrieved AVPairs related to the given document. Several processes are implemented to provide searching facility of term query, attribute query and term and attribute query. We implemented an extended database schema to stored document attributes created on the local peer.

We implemented a new GUI for the application to support all additional features. The new GUI allows user to insert a query with term or attribute, allow user to select merger or routing protocol, and also allow user to insert attribute of the document and distribute it to the system.

7.2 Future work

There are several possible improvements of our application. According to our design (PDI) that is mentioned in the chapter 4, our design still contains a drawback that the result of the global index may incomplete or even incorrect as mentioned in the chapter 4. We tried to resolve this drawback in DL and FDI. The DL approach suffers from hot spotting. The FDL inserts too much metadata into the global directory that may affect the searching efficiency of the global directory. One of the possible question for the future improvement is how to resolve this problem in such a cheap an effective way.

Bibliography

- [1] George Tzanetakis, Jun Gao, and Peter Steenkiste, *A scalable peer-to-peer system for music content and information retrieval*, Computer Music Journal, vol. 28:2, pp. 24-33, Summer, 2004.
- [2] Jun Gao, George Tzanetakis, and Peter Steenkiste, *Content-based retrieval of music in scalable peer-to-peer networks*, In Proc. of the Int.Conference on Multimedia and Expo, Jul 6, 2003-Jul 9, 2003.
- [3] Antony I. T. Rowstron and Peter Druschel, *Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems*, in Proc. of the 18th IFIP/ACM International Conference on Distributed System Platforms (Middleware 2001), Nov, 2001.
- [4] Antony Rowstron and Peter Druschel, *Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility*, In Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles, pp. 188-201, Oct, 2001.
- [5] Peter Druschel and Antony I. T. Rowstron, *PAST: A large-scale, persistent peer-to-peer storage utility*, In Proc. HOTOS Conf., 2001.

- [6] Peter Haase, Bjorn Schnizler, Jeen Broekstra, Marc Ehrig, Frank van Harmelen, Maarten Menken, Peter Mika, Michal Plechawski, Pawel Pyszlak and Ronny Siebes, Steffen Staab, and Christoph Tempich, *Bibster - A Semantics-Based Bibliographic Peer-to-Peer System*, In Proc. of the 3rd Int. Semantic Web Conference, Hiroshima, Japan, 2004.
- [7] Abhishek Gupta, *Attribute-based Data Access over P2P Systems*, Ph.D. Theses University of California, Santa Barbara, Sep, 2004.
- [8] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica, *Looking up data in P2P systems*, Communications of the ACM, vol. 2003.
- [9] Gero Muhl, *Large-Scale Content-Based Publish /Subscribe Systems*, PhD thesis, Darmstadt University of Technology, vol. 2002.
- [10] S. D. Gribble, A. Y. Halevy, Z. G. Ives, M. Rodrig, and D. Suciu, *What can database do for peer-to-peer?*, In Proc. Fourth International Workshop on the Web and Databases , vol. 2001.
- [11] M. Bender, S. Michel, C. Zimmer , and G. Weikum, *The MINERVA Project: Database Selection in the Context of P2P Search*, In: Datenbanksysteme in Business, Technologie und Web (BTW2005; 11. Fachtagung des GI-Fachbereichs Datenbanken und Informationssysteme (DBIS)), Karlsruhe, Germany, pp. 125-144 , 2005.

- [12] Bobby Bhattacharjee, Sudarshan S. Chawathe, Vijay Gopalakrishnan, Peter J. Keleher, and Bujor D. Silaghi, *Efficient peer-to-peer searches using result-caching*, In IPTPS, 2003.
- [13] Peter Haase, Nenad Stojanovic, Johanna Voelker, and York Sure, *Personalized Information Retrieval in Bibster, a Semantics-Based Bibliographic Peer-to-Peer System*, In Klaus Tochtermann and Hermann Maurer, Proceedings of the 5th International Conference on Knowledge Management (I-KNOW 05), pp. 104-111, Jul, 2005.
- [14] Peter R. Pietzuch and Jean Bacon, *Peer-to-Peer Overlay Broker Networks in an Event-Based Middleware*, In H. Arno Jacobsen, editor, Proceedings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS'03), ACM SIGMOD, vol. Jun, 2003.
- [15] Francisco Matias Cuenca-Acuna , Christopher Peery , Richard P. Martin , and Thu D. Nguyen, *PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities*, In Proc. Twelfth IEEE International Symposium on High Performance Distributed Computing (HPDC-12). IEEE Press, pp. 236-246, Jun, 2003.
- [16] J. LI, B. LOO, J. HELLERSTEIN, F. KAASHOEK, D. KARGER, and R . MORRIS, *The feasibility of peer-to-peer web indexing and search*, In 2nd International Workshop on Peer-to-Peer Systems, Berkeley, California, 2003.
- [17] P. Reynolds and A. Vahdat, *Efficient peer-to-peer keyword searching*, P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. Unpublished manuscript.

- [18] Rudiger Schollmeier, *A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications*, In Proc. of the First International Conference on Peer-to-Peer Computing (P2P '01), pp. 101-102, Aug, 2001.
- [19] A. Singh and M. Srivatsa and L. Liu and T. Miller, *Apoidea: A Decentralized Peer-to-Peer Architecture for Crawling the World Wide Web*, Lecture Notes in Computer Science, 2924, 2004,
- [20] T. Suel, C. Mathur, J. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasunderam, *Odissea: A peer-to-peer architecture for scalable web search and information retrieval*, 2003.
- [21] *Nepster*, <http://www.napster.com/>.
- [22] *Gnutella*, www.gnutella.org.
- [23] Ming Xie, *P2P Systems Based on Distributed Hash Table*, computer Science, University of Ottawa, September 26, 2003
- [24] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, *Chord: A scalable peer-to-peer lookup service for internet applications*, In Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM), ACM Press, 2001 pp. 149–160.

- [25] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, *A scalable content-addressable network*, in Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM), ACM Press, 2001, pp. 161–172.