An Architecture for Online Information Integration on Concurrent Resource Access on a Z39.50 Environment

Michalis Sfakakis¹ and Sarantos Kapidakis²

¹ National Documentation Centre / National Hellenic Research Foundation,
48 Vas. Constantinou, GR-11635 Athens, Greece
msfaka@ekt.gr

² Archive and Library Sciences Department / Ionian University,
Plateia Eleftherias, Paleo Anaktoro, Corfu 49100, Greece
sarantos@ionio.gr

Abstract. The lack of information integration, by the existing online systems, for resource sharing in a distributed environment, impacts directly to the development and the usage of dynamically defined Virtual Union Catalogues. In this work we propose a design approach for the construction of an online system, able to improve the information integration when a Dynamic Resource Collection is used, by taking into account the restrictions imposed by the network environment and the Z39.50 protocol. The main strength of this architecture is the presentation of de-duplicated results to the user, by the gradual application of the duplicate detection process in small received packets (sets of results), as the data packets flow from the participating servers. While it presents results to the user, it also processes a limited amount of data ahead of time, to be ready before the user requests them.

1 Introduction

The development of Union Catalogues has impacted resource sharing, giving the users the ability to access information, describing library content from one central point. The first implementations of Union Catalogs were based on the centralized model, due to technology limitations. In our days, the current technology, combined with the implementation of the Z39.50 protocol [1], enables the implementation of Union Catalogues based on the distributed model. Independently of the Union Catalogue underlining implementation model, the consistent searching and indexing, the consolidation of records and the performance and management are the main functional requirements that they have to conform to [7].

Until now, all of these functional requirements are satisfied well, in Union Catalogs implemented according to the centralized model, remaining open issues for virtual Union Catalogs, dynamically defined from distributed resources [2, 6, 7]. The local autonomy and control of the participating systems [7], plus the ability of the user to define his/her own virtual collections from a common interface [11], necessitates the design and the implementation of systems based on the distributed model.

The development and the implementation of the Z39.50 protocol gave the challenge to the library community, not only to have a standard model for their system development, but also to focus on higher levels of interoperability after solving the lowest level of system interoperability [8, 9]. Higher levels of interoperability include the consistent searching and indexing of the distributed resources and are mainly affected by the local implementation of standards and associated profiles of the Z39.50 protocol [3, 4, 10].

Networks, computer systems and interoperability improvements, give the base for approaches to design online, real time systems according to the distributed model. The current implementations of the systems that build dynamically defined virtual Union Catalogues, mix the result sets but do not merge them, by applying duplicate detection procedures, to avoid performance issues. When some implementations try to approach the problem, the consolidation of records is based on some coded data (e.g. ISBN, ISSN, LC number, etc.), when they exist. The result set merging (consolidation of records) is a requirement identified by user studies [11] and has to be improved by the systems proposed to support multi database searching in any environment, networked or not. A deep solution to this problem requires the retrieval of all the results, which probably could only be implemented in offline circumstances.

The problem of the information integration in a distributed environment is not a new one, and mainly appears in two variations. When applied in the development of centralized Union Catalogues or in collection creation by harvesting techniques, the challenge is to make high quality duplicate detection and merging on large amount of data offline, without serious time restrictions. In contrast, when applied in virtual Union Catalogues, dynamically defined from distributed resources, the challenge is to make good quality duplicate detection, on medium to small amount of data online and present them to the user in real time.

This work proposes a system architecture for online use, built on the distributed search model, for improving the consolidation of records without performance penalties, taking into account the Z39.50 protocol futures. The main strength of this architecture is the presentation of de-duplicated results to the user, by the gradual application of the duplicate detection process in small received packets (sets of results), as the data packets flow from the participating servers. While it presents results to the user, it also processes a limited amount of data ahead of time, to be ready before the user requests them.

In this work, we use the Z39.50 protocol because it is widely used in the area of libraries and also due to its complexity, the different levels of support for relevant facilities and the varieties of the existing implementations. Most of the proposed architecture is protocol-independent and could be applied equally well for other client/server information retrieval protocols, responding in real time, even if they provide partial results each time or they timeout on inactivity.

The rest of the paper organized as follows: section 2 describes the problems for the online information integration from concurrent resource access and the way that it works in a distributed Z39.50 environment. Section 3 describes the practices of the currently used systems. Section 4 describes the architecture of the proposed system and finally section 5 concludes the paper and presents a number of interesting issues arrived from this work for further development and research.

2 Problem Description

The Z39.50 protocol is based on the client/server model. The client (origin) sends an init request to the server (target) and if the server accepts the request, it starts a new session (a Z-association). The init respond, sent by the server to the client, includes information about the server characteristics (e.g. protocol version, supported services, etc.). For every Z-association, the server holds a search history (i.e. the result sets for the performed queries), which includes the last result set or more, depending on the support of the named result sets by the server. During the session, the client can request data from any result set included in the search history. The search history stays alive during the session, while either the origin or the target, via the close service, can abruptly terminate the session. The protocol defines that one 'close-reason' of the session is the 'lack of activity', which means that the server can close (timeout) the session when the client does not send a request for a period of time. The timeout period is server dependent.

The termination of the communication between the Z39.50 client and server costs the loss of the previously issued results and their arrangements (duplication detection or sorting). The client, in order to be able to continue the process from the termination point, has to reconnect to the timed out server, re-execute the query and finally request the rest of the data. Two problems are issued by the session reactivation: The first is the performance overhead. The second is that there is no guarantee that the new result set will be identical with the previously working one. This means that, in order to have consistent results, we have to retrieve all the data from the beginning of the result set and to reapply the whole process.

Depending on the implementation level of the Z39.50 protocol by the server, some servers could implement, in their resources, the duplicate detection service [12], the sort service or none of them. In a distributed concurrently accessed environment, this means that when a user defines a virtual collection of databases (resources), some databases could belong to a server which implements neither the duplicate detection service nor the sort service, which is the majority of the current Z39.50 implementations. Additional interoperability issues are the variations in the implementation level of the supported duplicate detection or sort service and the differences in the applied duplicate detection algorithms.

The duplicate detection problem occurs, in different forms, in a number of disciplines whenever the integration of information is required. In the area of libraries, from the late of 70s, during the development of centralized union catalogues, one approach to the problem was the automatic detection of duplicate monographic records [5]. The differences in semantic models and formats used for the metadata encoding, as well as their quality and consistency, are crucial issues affecting the overall effectiveness and the quality of the de-duplication process. Usually, after the identification of the related records by the duplicate detection process, the next step is the merging of the duplicates in a union record. According to the Z39.50 duplicate detection model [12], the duplicates are clustered in 'equivalence classes', by applying the matching criteria to a specific portion of each record. A representative record is selected for every equivalence class, which is also retained and arranged (sorted) by a combination of the values from some parameters.

A Z39.50 client which communicates concurrently with a number of disparate Z39.50 servers has to deal with the heterogeneity in the way it receives the resulting data (e.g. some results do not include duplicates, others are only sorted and others are received in a server dependent manner). Even at the best case, when the client receives de-duplicated results from all the servers, it also has to de-duplicate them. The client, also, has to take care of the differences between the timeouts and the performances of the participating servers. Finally the client has to perform all its operations and responds to the user in real time, or at least with limited delays.

Due to the complexity and the expensiveness of the duplicate detection algorithms, high-quality consolidation process for large amounts of records can be applied mostly in offline environments [7]. In an online environment (web based or not), where the user in most cases does not retrieve all his results at once but progressively in small packets, expecting the results promptly, we can only afford to apply quality de-duplication process on small number of records to create the sets of unique records to present. To do this, we have to invent heuristics to save processing time, even by sacrificing some accuracy in the duplicate detection process. One possible way to achieve this could be by applying the de-duplication process in the first waves of data sets that arrive from the servers, during the first times we present data to the user. This approach postpones the problem of comparing large amounts of records – if needed at all – to later steps, during the time the user is reading the presented data.

3 Practices of the Currently Used Systems

The majority of the current implementations of the Z39.50 servers support the basic services of the protocol (e.g. Init, search, present and scan). A small number of servers support the sort service and a minority of them supports the duplicate detection service. The lack of the implementation for the duplicate detection service is due to the late specification and acceptance of the service by the protocol and also due to its implementation complexity. This situation impacts to the efficiency of the clients, when they concurrently access multiple resources from different Z39.50 servers.

At the Z39.50 client side, the current implementations, when they give the ability to the user to access multiple resources, they work dispatching the requests from the user, sequentially or in parallel, to the participating systems, and after receiving the responses, they start to presenting the results sequentially back to the user. At the first presentation step, some systems, usually the web based, display a specific number of results from every resource, or a specific number of results from one resource (the first selected). After the first presentation step, they also give the ability to the user to request results from a specific resource. Other systems start downloading the results sequentially from each server. The user is able to read the received data, but the client has to download all the results in order to accomplish the duplicate detection process. The last approach is forbidden in online interactive environments, especially for large amounts of data, due to response time and timeout issues.

In the most common case, there is not uniformity in the structure and the content of the received result sets from the servers. Finally, in any case, the problem of the de-duplication and integration of the information from all the resources, which depends

on the client's functionality, may involve less or more work, but it exists. Due to performance issues, the majority of the existing clients do not make any integration of the received information, by a duplicate detection process, or they make primitive approaches based on some coded data (e.g. ISBN, ISSN, LC number, etc.).

4 System's Architecture Description

In a distributed resource environment, we have a set R of r participating resources (i.e. $r = |R|$), available from a set S of s servers (i.e. $s = |S|$). Every server from S could be accessed remotely via the network and could publish more than one resource from R . When the retrieval of the distributed resources accomplished by an online system, the system displays the results in steps, by generating *Presentation Sets* with a small number of results. The maximum number p of the results that every *Presentation Set* P will contain is defined by negotiation of the user and the system.

The user interacts with the system by first defining the desired collection of resources. This action, in a more advanced system, could be automatically accomplished by the system, using the appropriate metadata describing the content of the resources, or by both in cooperation. Then the user sends a search request. He could also specify the desired number of records (e.g. 25), which the system will include in each *Presentation Set* response. If the number of records, which satisfies the user's request, is greater than the defined number of records for display, then the user could subsequently request the remaining *Presentation Sets*.

Two major critical points coexist in an online distributed system, which accesses concurrently resources via the network and integrates the information, by applying duplicate detection procedures for the results returned by the participating systems. The first is the performance of the network links and the availability of the resources. The second is the complexity and the expensiveness of the duplicate detection algorithms, for presenting to the user only once each record that is multiply located in the resources.

In our proposed system, the duplicate detection process is applied using each received set of data from the servers and comparing them against the previously processed results - which are sorted and do not contain duplicates. We incrementally compare and eliminate the duplicate records in every metadata record presentation. When it is possible, we use the appropriate services supported by the servers (e.g. duplicate detection or sort) in order to get data in more processed condition and consequently to apply more specialized and efficient algorithms. Also, during the time the user is reading a set of results, the system prepares the next sets of unique records for display, by requesting a small portion of the remaining data from the servers and applying the duplicate detection process. We do not apply the duplicate detection algorithm in one shot, to all metadata records that belong to a specific query result. The benefits of this approach are that we avoid downloading large amounts of data from the participating systems over the network and we apply the duplicate detection algorithm to a small number of records in the first steps.

The system consists of the *Request Interface*, the *Data Integrator* and the *Resource Communicator* modules. The *Data Integrator* is composed by a number of interacting

components, in order to accomplish the information integration for the data arrived from the *Resource Communicator* module. The interaction between the modules and the components is accomplished either by synchronous data transmissions using queues or by messages. Fig. 1 outlines System's modules, components and their interactions. The rectangles represent the modules, the ellipses represent the components, the solid lines represent data transmissions and the dashed lines represent control message transmissions. The ellipsis with the double lines indicates that, in an expanded architecture, multiple instances of the component could run on different CPUs and communicate by shared memory, for increasing the overall response time.

Fig. 1. System's modules, components and their interactions

4.1 Modules, Components, and Interactions

The three modules from which the proposed system is composed of are the *Request Interface*, the *Data Integrator* and the *Resource Communicator*. Their description and their interactions are:

- The *Request Interface* receives every user request (search or present) and dispatches it to the appropriate internal modules, waiting for (at most p) records (i.e. the *Presentation Set*) to display. When the module receives the de-duplicated results from the *Data Integrator* module, it sends them to the user and waits for further commands.
- The *Resource Communicator* accesses the resources, using the appropriate communication protocols (e.g. Z39.50), and sends them the appropriate commands to retrieve data from them, according to the user request, which is sent by the *Request*

Interface module. It also takes into account the services that the servers support and their differences in the timeout and the performance, from the *Profiles of the Z39.50 Servers*. If a server supports the duplicate detection or the sort service, it requests from the server to use them. The module does not retrieve all the resulting records from every resource, but every time it only retrieves at most p (actually, there can be a different p per resource, but to simplify the description we assume it is the same, and is the number of records the *Request Interface* waits for displaying). By this approach, the *Resource Communicator* avoids downloading vast amounts of data from the network and in every ‘request data step’ it retrieves at most p records from every resource. When the module receives a result set from a resource, it sends it to the *Data Integrator* module for the de-duplication process. The *Resource Communicator* starts its action when it receives a search request by the *Request Interface* module and sends messages to it when access control is in effect by a server or a resource, or when a server is not available.

- The *Data Integrator* receives the data sets from the *Resource Communicator*, makes the information integration by applying the duplicate detection process and manages the processed unique records to be ready for presentation to the user. When the first *Presentation Set* with non-duplicate records is accomplished, it sends it to the *Request Interface*. Subsequently, it prepares the next *Presentation Set* and waits for commands for possibly different sets from the *Request Interface*.

All the components or modules of the system accomplish their actions working in parallel, unless they wait for input (data or control message transmission) from another component or module.

The *Data Integrator* module accomplishes its goal by the interaction of the four internal components, *Data Provider*, *Local Result Set Manager*, *De-duplicator* and *Data Presenter*. Their description is:

- The *Data Provider* receives data from the *Resource Communicator* module and subsequently sends them, one at a time, to the appropriate component, which will further process them. The *Data Provider* is using internally two queue structures and guarantees to be able to receive data from the *Resource Communicator* at any time, even without prior request and also to always provide enough data for processing. The first task ensures that the *Resource Communicator* will be able to retrieve data from the resources in emergency conditions (i.e. some servers are close to timeout their connection), or due to slow performance from any server. For the control of the second task, when the number of data contained in the *Data Provider* is less than a threshold, the *Data Provider* sends a ‘request data’ message to the *Resource Communicator*.
- The *Local Result Set Manager* holds and arranges (e.g. sorts) the de-duplicated records in its internal *Local Result Set* structure. The first task of the *Local Result Set Manager* is the activation of the duplicate detection component (i.e. *De-duplicator*) for all the incoming records. It also prepares the *Presentation Set*, using the presentation set queue, by copying the p records (i.e. the number of requested records for display) from the *Local Result Set* structure. When the number of the existing records in the *Local Result Set* is less than p , an ‘end of data’ is appended into the *Presentation Set* queue.
- The *De-duplicator* receives a record from the *Local Result Set Manager* and accomplishes the duplicate detection process. It compares the record with all the rec-

ords, appropriately ordered, in the *Local Result Set* Structure. After the comparison process, it sends the results from the process (i.e. the set of records that this record belongs to, or none, if the record does not match with any set) to the *Local Result Set Manager*, which then places the record in the right position in the *Local Result Set*. The *Local Result Set Manager* sends all incoming records to the *Duplicator*.

- The *Data Presenter* is used for the interaction with the *Request Interface* module. It dispatches the received requests for data, from the *Request Interface*, to the *Local Result Set Manager* and returns the unique records (if any) back to the *Request Interface* module. When the *Data Presenter* dispatches the present request for p records to the *Local Result Set Manager*, it waits (blocked by reading) for the Presentation Set queue to be filled with the requested number of records. Finally, it also monitors the system timeout on the connection with the user (i.e. no request for data from the *Request Interface* module for a period of time), closing the session.

4.2 Accomplishing a Search Request

The system starts its action when the user submits a search request and the *Request Interface* module receives the request. The following steps describe the functions, the flow of data and control messages transmissions performed by the modules of the system.

When the *Request Interface* module receives a search request, from the user interface subsystem, the following actions take place:

- 1 The *Request Interface* module sends a 'request data' message for p records to the *Data Integrator* module and then waits for (at most p) records from it.
- 2 The *Request Interface* module, also, forwards the search request including the number p , to the *Resource Communicator* module and continues monitoring for user requests.
- 3 The *Resource Communicator* module waits for messages from the *Request Interface* and when it receives a new search request, it concurrently starts the following sequences of steps for every server:
 - 3.1 The *Resource Communicator* module interprets the search request to the appropriate message format for the server, sends it to the server and waits for its reply.
 - 3.2 When the reply is received from the server, it adds the number of records (hits) found, for the user request in the server's resources. If the replies from all the servers have arrived, it sends the sum to the *Request Interface* module.
 - 3.3 If the server supports either the duplicate detection service or the sort service, it also request to de-duplicate or sort its results, after its initial response to the search request.
 - 3.4 Requests a number of records (e.g. p) from every server, which replied on its last request.
 - 3.5 When a result set arrives, it sends the data to the *Data Integrator module*.
 - 3.6 It waits for further commands, but if there is no communication with the server for a period close to its timeout, the procedure jumps to step 3.4.

- 4 The *Data Integrator* module de-duplicates part of the data sent by the *Resource Communicator* and prepares the *Presentation Set*. When p unique records are found, the module sends them to the *Request Interface* module. If at any time the number of records available for de-duplication is less than a threshold (e.g. $5p$), the *Data Integrator* module sends a 'request data' message to the *Resource Communicator* module, which subsequently repeats the steps 3.4 and 3.5 for each server.

Here are some comments and clarifications for the steps described previously. The *Request Interface* module sends the requested number p to the *Resource Communicator* module in order to make known how many records (how big the *Presentation Set* is) the user expects to retrieve from every system response. The *Resource Communicator* module can use it to calculate the number of records that will request from the servers (step 3.4). The number of requested records could be different for every server and could be greater or less than p . The *Resource Communicator* module calculates the number of records to request, using the characteristics for every server, from the *Profiles of the Z39.50 servers*, and the number of results found in every resource. When a server has slow performance the number of requested records could vary to avoid long waiting time. Also, the number of requested records could be small, when the server has a short timeout period. In this case the *Resource Communicator* avoids the session reactivation by often requesting data from it. In contrast, the number of requested records could be big, when the number of results is big and the server has fast performance and fast network links.

The *Resource Communicator* starts its action when it receives a search request by the *Request Interface* module (step 3) and sends messages to it if access control is in effect by a server or a resource. The message could be sent to the *Request Interface* module before contacting the server (step 3.1), if the *Profiles of the Z39.50 servers* contain the appropriate information, or after the step when a server or a resource is challenged by access control. Also, the *Resource Communicator* informs the *Request Interface* when a server is down. If the *Resource Communicator* knows that the server will not be available for a period of time, from the *Profiles of the Z39.50 servers*, the module sends the message before the step 3.1. In any other case it sends the message after this step. A critical decision of the *Resource Communicator*, for the overall system performance, is to understand that a server is down and to continue the interaction with the other modules, with the results from the other available resources. For this decision, the *Profiles of Z39.50 servers* could help if they contain information about the response time of each server and the unavailability history.

At step 3.2, the *Resource Communicator* module calculates the number of hits found for the user's request and returns it to the *Request Interface* module (which is performed once for every search request). This number is not actually accurate, because the system has not accomplished yet the record de-duplication. This information is very essential to the user (and is expected by the Z39.50 protocol), in order to decide for an alternative, more specific request. Even at the best case, when all servers support the duplicate detection service, this number is only somehow closer to the real one. The only way to calculate the precise number of hits found by the servers is to retrieve and de-duplicate all the data from all the resources, which is forbidden in on-line environments

The sequence of steps from 3.1 through 3.6, used for search and retrieve the resources of a server, is performed in parallel with the corresponding steps of other

servers. Also the *Resource Communicator* module could skip step 3.3 and go directly to step 3.4, for a server that supports neither the duplicate detection nor the sort service.

The *Resource Communicator* module, in order to avoid the session reactivation, which is imposed by the existence of the timeout from a server, could request data from a resource, at any time, independently of the existence of a 'request data' message. The module derives the information concerning the timeout period of a server from the *Profiles of the Z39.50 servers*.

Finally at step 4, where the *Data Integrator* module prepares the *Presentation Set*, the module could request data from the *Resource Communicator* module, not only when the number of current de-duplicated results is less than p , but also when it is less than a threshold. This action utilises the time the user is reading the data and the system prepares the data for the next *Presentation Set*.

4.2.1 Data Integrator Internal Description

After the explanation of the search request execution progress, we further analyze how the internal components of the *Data Integrator* module perform the functions, the data transmissions and the flow of control message transmissions, when the module is activated, as described in steps 1 and 4 at section 4.2. The *Data Integrator* module consists of four internal components, as we already described, and interoperates with both the other modules. The first message that the *Data Integrator* receives, through its *Data Presenter* component, comes from the *Request Interface* module (step 1). The *Data Presenter* instructs all components to flush any data from previous requests and informs the *Local Result Set Manager* that it is expecting p records. Initially all the components in the module are in an idle state, waiting for data to process. Regardless of this, the module actually starts its action immediately after the first record written, by the *Resource Communicator* module (step 3.5), to the *Data Provider* component. From this point, the *Data Integrator* module starts working in parallel with the other modules.

When the *Resource Communicator* module writes the first record to the *Data Provider*, the following actions take place inside the *Data Integrator* module:

- 1 The *Data Provider* starts to transfer data from its input queue to its output queue possibly by rearranging them, for example in an order that optimizes the initial processing time. If the number of data contained in it is less than a threshold (e.g. $5p$), the *Data Provider* sends a 'request data' message to the *Resource Communicator*.
- 2 While the *Local Result Set Manager* has less than a threshold (e.g. $3p$) unique record, it tries to read from the *Data Provider* and for every record found, it calls the *De-Duplicator* to compare the record.
 - 2.1 The *De-Duplicator* compares the record with the records in the *Local Result Set* and then sends the results back to the *Local Result Set Manager*.
 - 2.2 The *Local Result Set Manager* receives the results from the duplicate detection process and arranges the record into the *Local Result Set*.
 - 2.3 If the number of new unique records in the *Local Result Set* becomes p , it copies the p new unique records into the *Presentation Set* and activates the *Data Presenter*.

3 When the *Presentation Set* is filled with (the p) records, the *Data Presenter* component dispatches the records to the *Request Interface* module and waits to receive the next ‘request data’ message from it. If the component does not receive any request during its predefined timeout period, it terminates the system.

As we have already reported, every component works in parallel with the others, when it requires no input from another component.

One of the activities of the *Data Integrator* module is to request data from the *Re-course Communicator*, when the already retrieved data are not enough to complete the *Presentation Set* and to prepare at least the next one. On the other hand, when enough records (for the next possible request) have been de-duplicated, some components of the *Data Integrator* will be in an idle state mode, releasing the CPU and avoiding downloading more records from the resources with no apparent reason (e.g. if a server is close to timeout the connection). This behavior is affected by two thresholds; the first concerns the number of records in the *Data Provider* and the second concerns the number of the de-duplicated records in the *Local Result Set* which makes the *Local Result Set Manager* to stop reading records from the *Data Provider*, leaving the *De-duplicator* inactive.

The *Local Result Set Manager* keeps two orderings of the records in the *Local Result Set* (step 2.2). The first, sorts the records in order to support the *De-Duplicator* to apply duplicate detection algorithms efficiently. The second arrangement facilitates easy location of the non-presented records to the user in a sorted way. This ordering is different, especially if some newly de-duplicated unique records are placed before elements already presented to the user.

4.3 Accomplishing a Present Request

As we described in section 4.2, the system starts its action when the user submits a search request. After this, the user could request from the system to retrieve another *Presentation Set*. In this case, the *Data Presenter* requests the records from the *Local Result Set Manager* in the *Data Integrator* module, and waits until the *Data Integrator* has found the requested records, performing the steps of section 4.2.1. When the requested number of records is available in the *Local Result Set*, it fills the *Presentation Set* with the records and performs step 3 of section 4.2.1.

5 Conclusions and Future Research

The online duplicate detection process (as a part of the information integration) from resources accessed concurrently in a network environment, is a requirement identified by user studies and is challenged by a number of issues relevant to the performance of the participating servers and their network links, plus to the complexity and the expensiveness of the duplicate detection algorithms. These issues make inefficient any approach to the application of the information integration in online environments, especially when large amounts of data must be processed. In our system we do not try to integrate all the results from all the recourses at once. We attack this problem by retrieving a small number of records from the participating resources (independently if they provide de-duplicated or sorted results), we apply the duplicate detection process on these records and we create a presentation set of unique records to display to the

user. In cases where the user query results to a large amount of records, he usually does not retrieve all the results anyway, but reforms its query.

From this work, a number of interesting points arrives for future development and research. A better approximation for the number of records satisfying the search request, possibly using the Z39.50 facilities and preview statistics, is a critical point for assisting the user to decide how more specific he must be when re-expressing his request. Another point of interest is how to extend the system to derive priorities for the servers and their resources, based on supported features, performance statistics from previous accesses and other parameters, and at the same time to take into account user defined priorities possibly based on the importance of the content of the resources. Another interesting problem is a selection or an adaptation to a good de-duplication algorithm for different record completeness and different provision of records by the servers (e.g. all sorted, none sorted, all de-duplicated, none de-duplicated, etc.). Also, an approximation method for the number of requested records from a server, using its characteristics (e.g. performance) and the number of results found in its resources is another issue to solve. Finally an implementation of the proposed system and its performance evaluation will show points for further system improvement and extensions for other services.

References

1. ANSI/NISO: Z39.50 Information Retrieval: application service definition and protocol specification: approved May 10, 1995.
2. Coyle, Karen. The Virtual Union Catalogue: A Comparative Study. D-Lib Magazine vol. 6:num. 3 (March 2000).
3. Dunsire, Gordon. Joined up indexes: interoperability issues in Z39.50 networks. 68th IFLA Council and General Conference, August 18–24, 2002.
4. Gill, Tony and Miller Pall. Re-inventing the Wheel? Standards, Interoperability and Digital Cultural Content. D-Lib Magazine vol. 8:num. 1 (January 2002)
5. Hickey, Thomas B. and Rypka, David J. Automatic detection of duplicate monographic records. Journal of Library Automation, 12:2 (June 1979), pp. 125–142.
6. Lunau, Carrol & Turner, Fay. Issues Related to the Use of Z39.50 to Emulate a Centralized Union Catalogue. Prepared for the ARL Access Committee, National Library of Canada, April 1997. <http://www.nlc-bnc.ca/resource/vcuc/ezarl2.htm>.
7. Lynch, Clifford A. Building the Infrastructure of Resource Sharing: Union Catalogs, Distributed Search, and Cross-Database Linkage. Library Trends 45:3 (Winter 1997), pp. 448–461.
8. Lynch, C., Garcia-Molina H., “Interoperability, Scaling, and the Digital Libraries Research Agenda: A Report on the May 18-19, 1995”, IITA Digital Libraries Workshop, August 22, 1995. <http://www.diglib.stanford.edu/diglib/pub/reports/iita-dlw/>.
9. Moen, W. E. Mapping the Interoperability Landscape for the Networked Information Retrieval, JCDL 2001, June 24–28, 2001, Roanoke, Virginia, USA.
10. Moen, W. E. Improving Z39.50 Interoperability: Z39.50 Profiles and Testbeds for Library Applications. 67th IFLA Council and General Conference, August 16–25, 2001.
11. Payette, S. D. and Rieger, O.Y. “Z39.50 the User’s Perspective”, D-Lib Magazine, April 1997.
12. 13. Z39.50 Duplicate Detection Service (adopted by the Z39.50–1995 Amendment 2). <http://lcweb.loc.gov/z3950/agency/amend/dedup.html>