Information Retrieval Using a Middleware Approach

Danijela Boberić Krstićev

ABSTRACT

This paper explores the use of a mediator/wrapper approach to enable the search of an existing library management system using different information retrieval protocols. It proposes an architecture for a software component that will act as an intermediary between the library system and search services. It provides an overview of different approaches to add Z39.50 and Search/Retrieval via URL (SRU) functionality using a middleware approach that is implemented on the BISIS library management system. That wrapper performs transformation of Contextual Query Language (CQL) into Lucene query language. The primary aim of this software component is to enable search and retrieval of bibliographic records using the SRU and Z39.50 protocols, but the proposed architecture of the software components is also suitable for inclusion of the existing library management system into a library portal. The software component provides a single interface to server-side protocols for search and retrieval of records. Additional protocols could be used. This paper provides practical demonstration of interest to developers of library management systems and those who are trying to use open-source solutions to make their local catalog accessible to other systems.

INTRODUCTION

Information technologies are changing and developing very quickly, forcing continual adjustment of business processes to leverage the new trends. These changes affect all spheres of society, including libraries. There is a need to add new functionality to existing systems in ways that are cost effective and do not require major redevelopment of systems that have achieved a reasonable level of maturity and robustness. This paper describes how to extend an existing library management system with new functionality supporting easy sharing of bibliographic information with other library management systems.

One of the core services of library management systems is support for shared cataloging. This service consists of the following activities: a librarian when processing a new bibliographical unit first checks whether the bibliographic unit has already been recorded in another library in the world. If it is found, then the librarian stores that electronic records to his/her local database of bibliographic records. In order to enable those activities, it is necessary that standard way of communication between different library management systems exists. Currently, the well-known standards in this area are Z39.50[1] and SRU.[2]

Danijela Boberić Krstićev (dboberic@uns.ac.rs) is a member Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Serbia.

In this paper, a software component that integrates services for retrieval bibliographic records using the Z39.50 and SRU standard is described. The main purpose of that component is to encapsulate server sides of the appropriate protocols and to provide a unique interface for communication with the existing library management system. The same interface may be used regardless of which protocols are used for communication with the library management system. In addition, the software component acts as an intermediary between two different library management systems. The main advantage of the component is that it is independent of library management system with which it communicates. Also, the component could be extended with new search and retrieval protocols. By using the component, the functionality of existing library management systems would be improved and redevelopment of the existing system would not be necessary. It means that the existing library management system would just need to provide an interface for communication with that component. That interface can even be implemented as an XML web service.

Standards Used for Search and Retrieval

The Z39.50 standard was one of the first standards that defined a set of services to search for and retrieve data. The standard is an abstract model that defines communication between the client and server and does not go into details of implementation of the client or server. The model defines abstract prefixes used for search that do not depend on the implementation of the underlying system. It also defines the format in which data can be exchanged. The Z39.50 standard defines query language type-1, which is required when implementing this standard.

The Z39.50 standard has certain drawbacks that new generation of standards, like SRU, is trying to overcome. SRU tries to keep functionality defined by Z39.50 standard, but to allow its implementation using current technologies. One of the main advantages of the SRU protocol, as opposed to Z39.50, is that it allows messages to be exchanged in a form of XML documents, which was not the case with the Z39.50 protocol. The query language used in SRU is called Contextual Query Language (CQL).[3]

The SRU standard has two implementations, one in which search and retrieval is done by sending messages via the HyperText Transfer Protocol (HTTP) GET and POST methods (SRU version) and the other for sending messages using the Simple Object Access Protocol (SOAP) (SRW version). The main difference between SRU and SRW is in the way of sending messages.[4] The SRW version of the protocol packs messages in the SOAP Envelope element, while the SRU version of the protocol sends messages based on parameter/value pairs that are included in the URL. Another difference between the two versions is that the SRU protocol for messages transfer uses only HTTP, while SRW, in can use Secure Shell (SSH) and Simple Mail Transfer Protocol (SMTP), in addition to HTTP.

RELATED WORK

A common approach for adding SRU support to library systems, most of which already support, the Z39.50 search protocol,[5] has been to use existing software architecture that supports the Z39.50 protocol. Simultaneously supporting both protocols is very important because individual libraries will not decide to move to the new protocol until it is widely adopted within the library community.

One approach in the implementation of a system for retrieval of data using both protocols is to create two independent server-side components for Z39.50 and SRU, where both software components access a single database. This approach involves creating a server implementation from the scratch without the utilization of existing architectures, which could be considered a disadvantage.

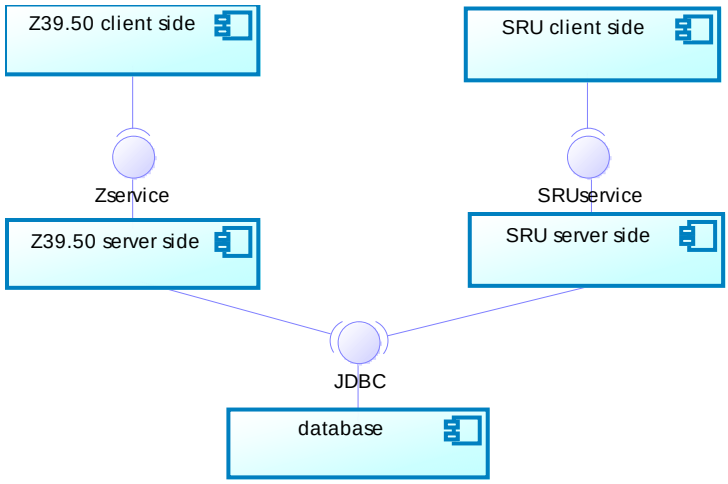

Figure 1. Software Architecture of a System with Separate Implementations of Server-Side Protocols

This approach is good if there is an existing Z39.50 or SRU server-side implementation, or if there is a library management system, for example, that supports just the Z39.50 protocol, but has open programming code and allows changes that would allow the development of an SRU service.

The system architecture that is based on this approach is shown in Figure 1 as a Unified Modeling Language (UML) component diagram. In this figure, the software components that constitute the implementation of the client and the server side for each individual protocol are clearly separated, while the database is shared. The main disadvantage of this approach is that adding support for new search and retrieval protocols requires the transformation of the query language supported by that new protocol into the query language of target system. For example, if the existing library management system uses a relational database to store bibliographic records, for every a new protocol added, its query language must be transformed into the Structured Query Language (SQL) supported by the database.

However, in most commercial library management systems that support server-side Z39.50, local development and maintenance of additional services may not be possible due to the closed nature of the systems. One of the solutions in this case would be to create a so-called "gateway" software component that implements both an SRU server and a Z39.50 client, used to access the existing Z39.50 server. That is, if a SRU client's application sends search request, the gateway will accept that request, transform it into the Z39.50 request and forward the request to the Z39.50 server. Similarly, when the gateway receives a response from the Z39.50 server, the gateway will transform this response in SRU response and forward it to the client. In this way, the client will have the impression that communicates directly with the SRU server, while the existing Z39.50 server will think that it sends response directly to the Z39.50 client. Figure 2 presents a component diagram that represents the architecture of the system that is based on this approach.

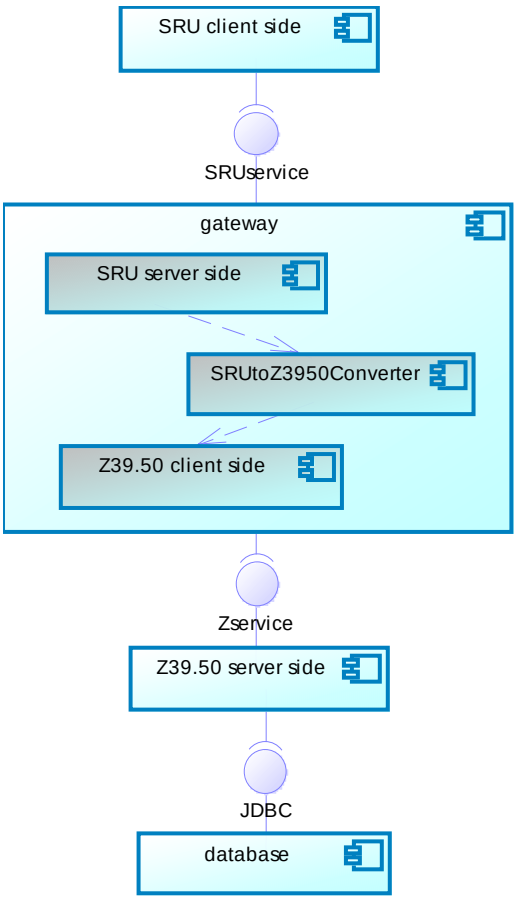

Figure 2. Software Architecture of a System with a Gateway

The software architecture shown in the Figure 2 is one of the most common approaches and is used by the Library of Congress (LC),[6] which uses the commercial Voyager[7] library information system, which allows searching by the Z39.50 protocol. In order to support search of the LC database using SRU, IndexData[8] developed the YazProxy software component,[9] which is an SRU-Z39.50 gateway. The same idea[10] was used in the implementation of the "The European Library"[11]

portal, which aims to provide integrated access to the major collections of all the European national libraries.

Another interesting approach in designing software architecture for systems dealing with retrieval of information can be observed in the systems involved in searching heterogeneous information sources. The architecture of these systems is shown in Figure 3.

The basic idea in most of these systems is to provide the user with a single interface to search different systems. This means that there is a separate component that will accept a user query and transform it into a query that is supported by the specific system component that offers search and data retrieval. This component is also known as a mediator. A separate wrapper component must be created for each system to be searched, to convert the user's query to a query that is understood by the particular target system.[12]

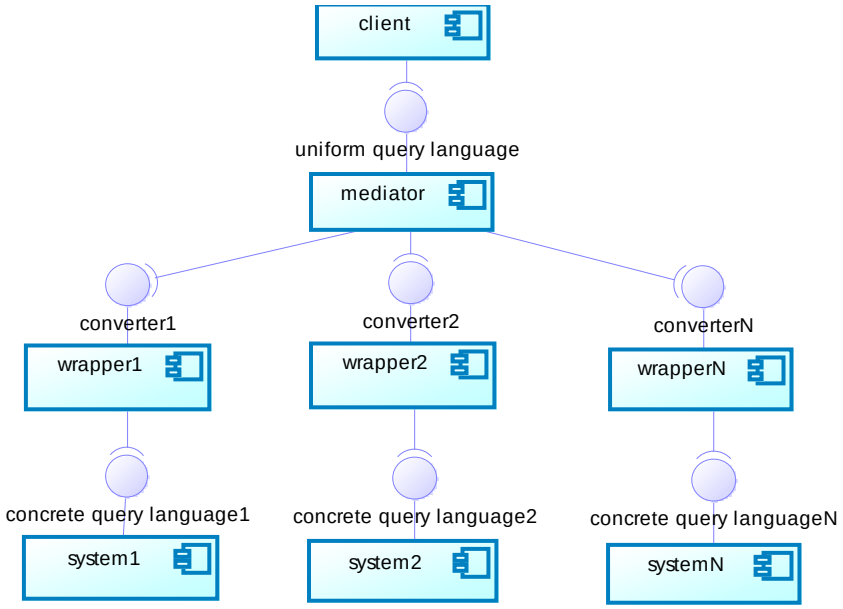

Figure 3. Architecture with the Mediator/Wrapper Approach

Figure 3 shows a system architecture that enables communication with three different systems (system1, system2 and systemN), each of which may use a different query language and therefore need different wrapper components (wrapper1, wrapper2 and wrapperN). In this architecture, each system can be a new mediator component that will interact with other systems. That is, the wrapper component can communicate with the system or with another mediator.

The role of the mediator is to accept the request defined by the user and send it to all wrapper components. The wrapper components know how to transform the query that is sent by a mediator into a query that is supported by the target system with which the wrapper communicates. In addition, the wrapper has to transform data received from the target system in a format prescribed by the mediator. Communication between client applications and the mediator

may be through one of the protocols for search and retrieval of information, for example through the SRU or Z39.50 protocols, or it may be a standard HTTP protocol.

Systems in which the architecture is based on the mediator/wrapper approach are described in several papers. Coiera *et al* (2005)[13] describe the architecture of a system that deals with the federated search of journals in the field of medicine, using the internal query language Unified Query Language (UQL). For each information source with which the system communicates, a wrapper was developed to translate queries from UQL into the native query language of the source. The wrapper also has the task of returning search results to the mediator. Those results are returned as an XML document, with a defined internal format called a Unified Response Language (UReL). As an alternative to using particular defined languages (UQL and UReL), a CQL query language and the SRU protocol could be used.

Another example of the use of mediators is described by Cousins and Sanders (2006),[14] who address the interoperability issues in cross-database access and suggest how to incorporate a virtual union catalogue into the wider information environment through the application of middleware, using the Z39.50 protocol to communicate with underlying sources.

Software Component for Services Integration

This paper describes a software component that would enable the integration of services for search and retrieval of bibliographic records into an existing library system. The main idea is that the component should be modular and flexible in order to allow the addition of new protocols for search and easy integration into the existing system. Based on the papers analyzed in the previous section, it was concluded that a mediator/wrapper approach would work best. The architecture of system that would include the component and that would allow search and retrieval of bibliographic records from other library systems is shown in Figure 4.

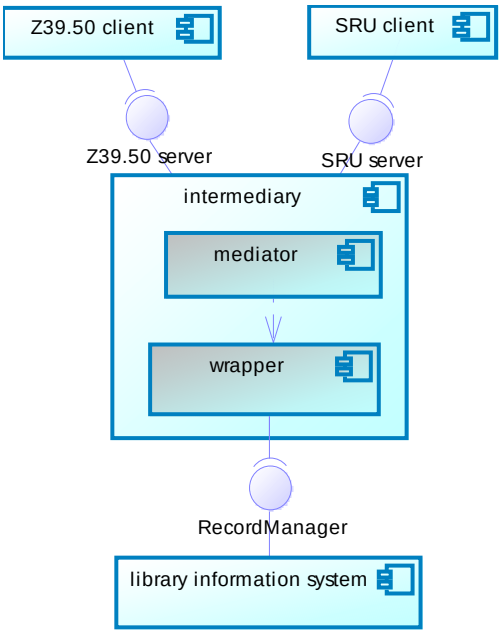

Figure 4. Architecture of System for Retrieval of Bibliographic Records

In Figure 4, the central place is occupied by the *intermediary* component, which consists of a *mediator* component and a *wrapper* component. This component is an intermediary between the search service and an existing library system. The library system provides an interface (*RecordManager*) which is responsible for returning records that match the received query. Figure 4 also shows the components that are client applications that use specific protocols for communication (SRU and Z39.50), as well as the components that represent the server-side implementation of appropriate protocols.

This paper will not describe the architecture of components that implement the server side of the Z39.50 and SRU protocols, primarily because there are already a lot of open-source solutions[15] that implement those components and can easily be connected with this *intermediary* component. In order to test the *intermediary* component, we used the server side of the Z39.50 protocol developed through the JAFER project[16] ; for the SRU server side, we developed a special web service in the Java programming language. In further discussion, it is assumed that the *intermediary* component receives queries from server-side Z39.50 and SRU services, and that this component does not contain any implementation of these protocols.

The mediator component, which is part of the intermediary component, must accept queries sent by the server-side search and retrieval services. The mediator component uses its own internal representation of queries, so it is therefore necessary to transform received queries into the appropriate internal representation. After that, the mediator will establish communication with the wrapper component, which is in charge of executing queries in existing library system. The basic role of the wrapper component is to transform queries received from the mediator into queries supported by library system. After executing the query, the wrapper sends search results as an XML document to the mediator. Before sending those results to server side of protocol, the mediator must transform those results into the format that was defined by the client.

Mediator software component

The mediator is a software component that provides a unique interface for different client applications. In this study, as shown in Figure 4, a slightly different solution was selected. Instead of the mediator communicating directly with the client application, which in the case of protocols for data exchange is client side of that protocol, it actually communicates with the server components that implement the appropriate protocols, and the client application exchanges messages with the corresponding server-side protocol. The Z39.50 client exchanges messages with the appropriate Z39.50 server, and it communicates with the mediator component. A similar process is done when communication is done using the SRU protocol. What is important to emphasize is that the Z39.50 and SRU servers communicate with the mediator through a unified user interface, represented in Figure 5 by class *MediatorService*. In this way the same method is used to submit the query and receive results, regardless of which protocol is used. That means

that our system becomes more scalable and that it is possible to add some new search and retrieval protocols without refactoring the mediator component.

Figure 5 shows the UML class diagram that describes the software mediator component. The *MediatorService* class is responsible for communication with the server-side Z39.50 and SRU protocols. This class accepts queries from the server side of protocols and returns bibliographic records in the format defined by the server.

The mediator can accept queries defined by different query languages. Its task is to transform these queries to an internal query language, which will be forwarded to the wrapper component. In this implementation, accepted queries are transformed into an object representation of CQL, as defined by the SRU standard. One of the reasons for choosing CQL is that concepts defined in the Z39.50 standard query language can be easily mapped to the corresponding concepts defined by CQL. CQL is semantically rich, so can be used to create various types of queries. Also, because it is based on the concept of *context set*, it is extensible and allows usage of various types of context sets for different purposes. So, CQL is not just limited to the function of searching bibliographic material. It could, for example, be used for searching geographical data. Accordingly, it was assumed that CQL is a general query language and that probably any query language could be transformed into it. In this implementation, the object model of CQL query defined in project CQL-Java[17] was used. In the case that there is a new query language, it would be necessary to perform mapping of the new query language into CQL or to extend the object model of CQL with new concepts.

This implementation of the mediator component could transform two different types of queries into the CQL object model. Currently, it can transform type-1 queries (used by Z39.50) and CQL queries into CQL object representation. To to add a new query language, it would just be necessary to add a new class that would implement the interface *QueryConverter* shown in Figure 5, but the architecture of component mediator remains the same.

One task of the mediator component is to return records in the format that was defined by the client that sent the request.

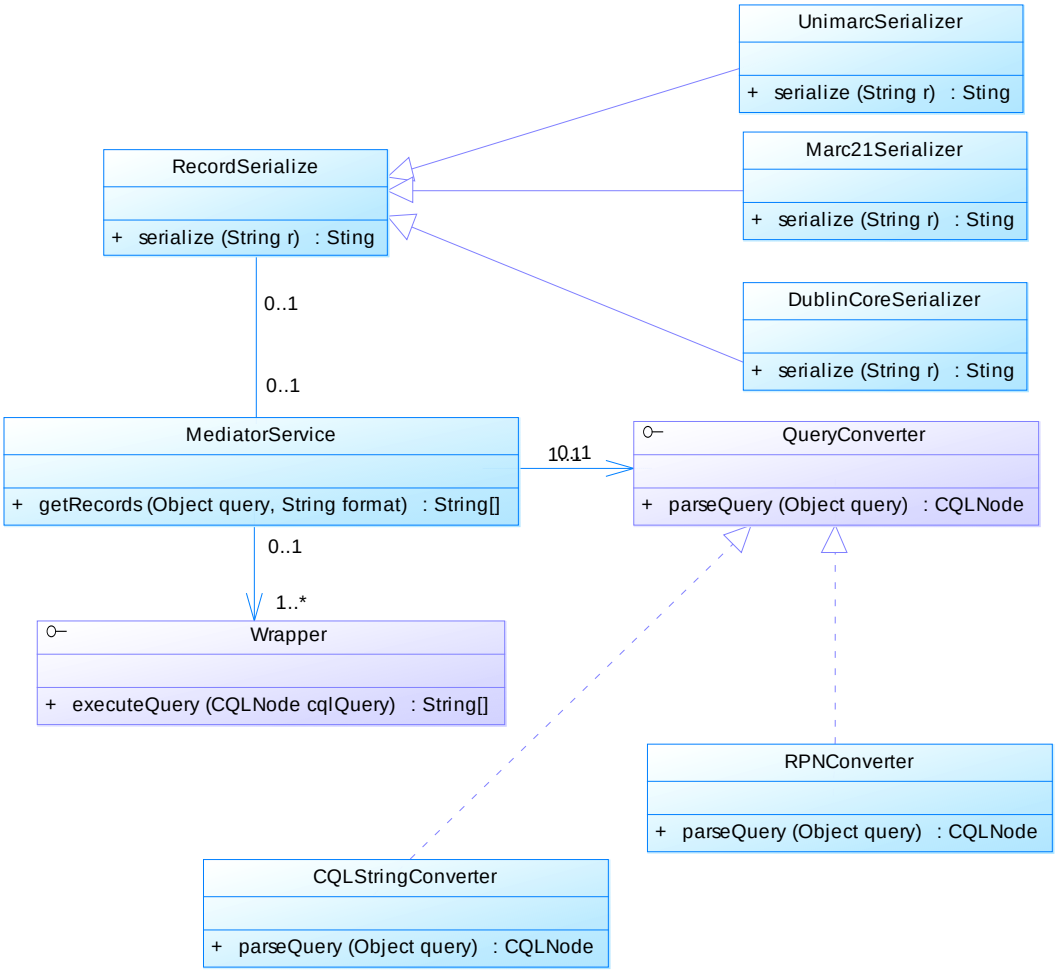

Figure 5. UML Class Diagram of Mediator Component

As the mediator communicates with the Z39.50 and SRU server side, the task of the Z39.50 and SRU server side will be to check whether the format that the client requires is supported by the underlying system. If it is not supported, the request is not sent to mediator. Otherwise, the mediator ensures the transformation of retrieved records into the chosen format. The mediator obtains bibliographic records from the wrapper in the form of an XML document that is valid according to the appropriate XML schema.[18] The XML schema allows the creation of an XML document describing bibliographic records according to the UNIMARC[19] or MARC21[20] format. The current implementation of the mediator component supports transformation of bibliographic records into an XML document that can be an instance of the UNIMARCslim XML schema,[21] the MARC21slim XML schema,[22] or the Dublin Core XML schema.[23] Adding support for a new format would require creating a new class that would extend the class *RecordSerializer* (Figure 5). Because this mediator component works with XML, the transformation of bibliographic records into a new format also could be done by using Exstensible Stylesheet Language Transformations (XSLT).

Wrapper software component

The wrapper software component is responsible for ensuring communication between the mediator and the existing library system. That is, the wrapper component is responsible for transforming the CQL object representation into a concrete query that is supported by the existing library system and for obtaining results that match the query. Implementation of the wrapper component directly depends on the architecture of the existing library system. Figure 7 proposes a possible architecture of the wrapper component. This proposed architecture assumes that the existing library system provides some kind of service that will be used by the wrapper component to send the query and obtain results. The *RecordManager* interface in Figure 7 is an example of such a service. *RecordManager* has two operations, one which executes the query and returns the number of hits and the second operation which returns bibliographic records. This proposed solution is useful for libraries that use a library management system that can be extended. It may not be appropriate for libraries using an "off the self" library management system that cannot be extended.

The proposed architecture of the wrapper component is based on a strategy design pattern,[24] primarily because of the need for transformation of the CQL query into a query that is supported by the library system. According to the CQL concept of context sets, all prefixes that can be searched are grouped in context sets, and these sets are registered with the Library of Congress. The concept of context sets enables specific communities and users to define their own prefixes, relations, and modifiers without fear that their name will be identical to the name of prefix defined in another set. That is, it is possible to define two prefixes with the same name, but they belong to different sets and therefore have different semantics.

CQL offers the possibility of combining in a single query elements that are defined in different context sets. When parsing a query, it is necessary to check which context set a particular item belongs to and then to apply appropriate mapping of the element from the context set to the corresponding element defined by the query language used in the library system.

The strategy design pattern includes patterns that describe the behavior of objects (behavioral patterns), which determine the responsibility of each object and the way in which objects communicate with each other. The main task of a strategy pattern is to enable easy adjustment of the algorithm that is applied by an object at runtime. Strategy pattern defines a family of algorithms, each of which is encapsulated in a single object. Figure 6 is shows a class diagram from the book "*Design Patterns: Elements of Reusable Object-Oriented Software,*"[25] which describes basic elements of strategy patterns.

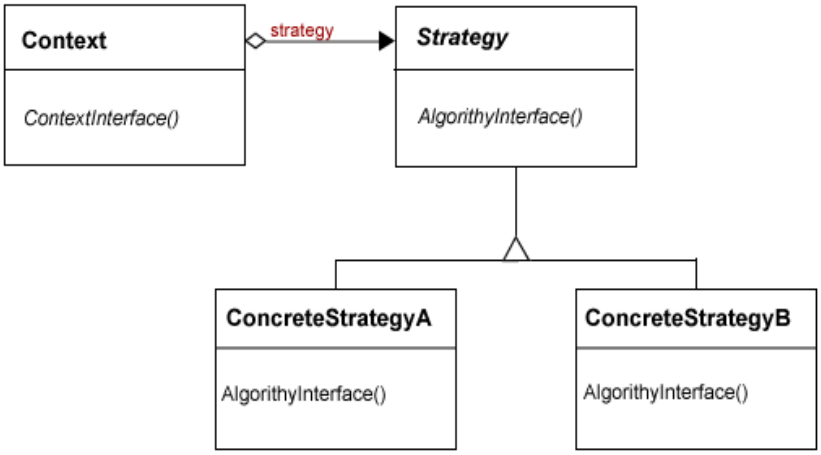

Figure 6. Strategy Design Pattern

The basic elements of this pattern are the classes *Context*, *Strategy*, *ConcreteStrategyA* and *ConcreteStrategyB*. The class *Context* is in charge of choosing and changing algorithms in a way that creates an instance of the appropriate class, which implements the interface *Strategy*. Interface *Strategy* contains the method *AlgorityInterface(),* which should implement all classes that implement that interface. Class *ConcreteStrategyA* implements one concrete algorithm.

This design pattern is used when transforming CQL queries primarily because CQL queries can consist of elements that belong to different context sets, whose elements are interpreted differently. Classes *Context*, *Strategy*, *CQLStrategy* and *DcStrategy*, shown in Figure 7, are elements of strategy pattern responsible for mapping concepts defined by CQL. The class *Context* is responsible for selection of appropriate strategies for parsing, depending on which context set the element that is going to be transformed belongs to. Class *CQLStrategy* and *DcStrategy* are responsible for mapping the elements belonging respectively to the CQL or Dublin Core context set in the appropriate elements of a particular query language used by the library system.

The use of strategy pattern makes it possible, in real time, to change the algorithm that will parse the query depending on what context set is used. The described implementation of a wrapper component enables the parsing of queries that contain only elements that belong to CQL and/or the Dublin Core context set. In order to provide support for a new context set, a new implementation of interface *Strategy* (Figure 7) would be required, including an algorithm to parse the elements defined by this new set.

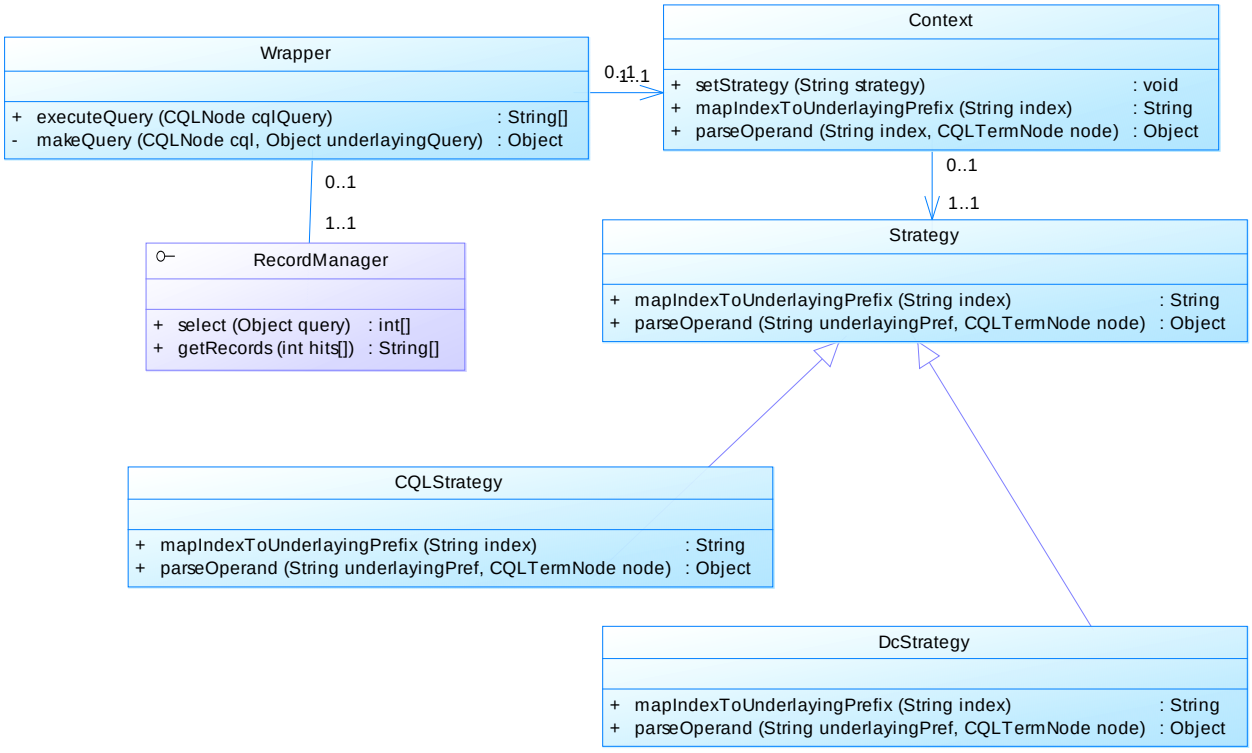

Figure 7. UML Class Diagram of Wrapper Component

Integration of Intermediary Software Components into the BISIS Library System

The BISIS library system was developed at the Faculty of Science and the Faculty of Technical Sciences in Novi Sad, Serbia, and has had several versions since its introduction in 1993. The fourth and current version of the system is based on XML technologies. Among the core functional units of BISIS[26] are:

- circulation of library material
- cataloging of bibliographic records
- indexing and retrieval of bibliographic records
- downloading bibliographic records through Z39.50 protocol
- creation of a card catalog
- creation of statistical reports

An intermediary software component has been integrated into the BISIS system. The intermediary component was written in the Java programming language and implemented as a web application. Communication between server applications that support the Z39.50 and SRU protocols and the intermediary component is done using the software package Hessian.[27] Hessian offers a simple implementation of two protocols to communicate with Web services, a binary protocol and its corresponding XML protocol, both of which rely on HTTP. Use of Hessian package makes it easy to create a Java servlet on the server side and proxy object on client-side, which will be used to

communicate with the servlet. In this case, the proxy object is deployed on the server side of protocol and the intermediary component contains a servlet. Communication between the intermediary and BISIS is also realized using the Hessian software package, which leads to the possibility of creating a distributed system because the existing library system, the intermediary component, and server applications that implement the protocols can be located on physically separate computers.

The BISIS library system uses the Lucene software package for indexing and searching. Lucene has defined its own query language,[29] so the wrapper component that is integrated into BISIS has to transform to the CQL query object model the object representation of the query defined by Lucene. Therefore the wrapper first needs to determine to which context set the index belongs and then apply the appropriate strategy for mapping the index. The rules for mapping the index to Lucene fields are read from the corresponding XML document that is defined for every context set. Listing 1 below provides an example of an XML document that contains some rules for mapping indexes of the Dublin Core context set to Lucene index fields. The XML element *index* represents the name of index which is going to be mapped, while the XML element *mappingElement* contains the name of Lucene field. For example, the title index defined in the DublinCore context set, which denotes search by title of the publication, is mapped to the field TI, which is used by the search engine of BISIS system.

```
<?xml version="1.0" encoding="UTF-8"?>
<contextSet identifier="info:srw/cql-context-set/1/dc-v1.1"  name="dc">
   <indexes>
      <index>
         <name>title</name>
         <mappingElement>TI</mappingElement>
      </index>
      <index>
         <name>creator</name>
         <mappingElement>AU</mappingElement>
      </index>
      <index>
         <name>subject</name>
         <mappingElement>SB</mappingElement>
      </index>
   </indexes>
</contextSet>
```

Listing 1. XML Document with Rules for Mapping the DublinCore Context Set

After the index is mapped to corresponding fields in Lucene, a similar procedure is repeated for a relationship that may belong to some other context set or may have modifiers that belong to some

other context set. It is therefore necessary to change the current strategy for mapping into a new one. By doing this, all elements of the CQL query are converted into a Lucene query, so the new query can be sent to BISIS to be executed.

Approximately 40 libraries in Serbia currently use the BISIS system, which includes a Z39.50 client, allowing the libraries to search the collections of other libraries that support communication through the Z39.50 protocol. By integrating the intermediary component in the BISIS system, non-BISIS libraries may now search the collections of libraries that use BISIS. As a first step, the intermediary component was just integrated in a few libraries, without any major problems. The component is most useful to the city libraries that use system BISIS, because they have many branches, which can now search and retrieve bibliographic records from their central libraries. The component could potentially be used by other library management system, assuming the presence of an appropriate wrapper component to transform CQL to the target query language.

CONCLUSION

This paper describes an independent, modular software component that enables the integration of a service for search and retrieval of bibliographic records into an existing library system. The software component provides a single interface to server-side protocols to search and retrieve records, and could be extended to support additional server-side protocols. The paper describes the communication of this component with Z39.50 and SRU servers.

The software component was developed for integration with the BISIS library system, but is an independent component that could be integrated in any other library system.

The proposed architecture of the software component is also suitable for inclusion of the existing library system into a single portal. The architecture of the portal should involve one mediator component whose task would be to communicate with wrapper components of individual library systems. Each library system would implement its own search and store functionalities and could function independently of the portal. The basic advantage of this architecture is that it is possible to include new library systems that provide search services. It is only necessary to add a new wrapper that will perform the appropriate transformation of the query obtained from the mediator component in a query that the library system can process. The task of the mediator is to send queries to the wrapper, while each wrapper can establish communication with a specific library system. After obtaining the results from underlying library system, the mediator should be able to combine results, remove duplicate, and sort results. In this way end user would have impression that he has been searched a single database.

REFERENCES

1. "Information Retrieval (Z39.50): Application Service Definition and Protocol Specification," http://www.loc.gov/z3950/agency/Z39-50-2003.pdf (accessed February 22, 2013).

2. "Search/Retrieval via URL," http://www.loc.gov/standards/sru/.

3. "Contextual Query Language – CQL," http://www.loc.gov/standards/sru/specs/cql.html.

4. Eric Lease Morgan, "An Introduction to the Search/Retrieve URL Service (SRU)," *Ariadne* 40 (2004), http://www.ariadne.ac.uk/issue40/morgan.

5. Larry E. Dixson, "YAZ Proxy Installation to Enhance Z39.50 Server Performance," *Library Hi Tech* 27, no. 2 (2009): 277-285, http://dx.doi.org/10.1108/07378830910968227; Mike Taylor and Adam Dickmeiss, "Delivering MARC/XML records from the library of congress catalogue using the open protocols SRW/U and Z39.50," (paper presented at World Library and Information Congress: 71st IFLA General Conference and Council, Oslo, 2005).

6. Mike Taylor and Adam Dickmeiss,"Delivering MARC/XML Records from the Library of Congress Catalogue Using the Open Protocols SRW/U and Z39.50," (paper presented at World Library and Information Congress: 71st IFLA General Conference and Council, Oslo, 2005).

7. "Voyager Integrated Library System," http://www.exlibrisgroup.com/category/Voyager.

8. "IndexData," http://www.indexdata.com/.

9. "YazProxy," http://www.indexdata.com/yazproxy.

10. Theo van Veen and Bill Oldroyd, "Search and Retrieval in The European Library," *D-Lib Magazine* 10, no. 2 (2004), http://www.dlib.org/dlib/february04/vanveen/02vanveen.html..

11. "The European Library," http://www.theeuropeanlibrary.org./tel4/.

12. Gio Wiederhold ,"Mediators in the Architecture of Future Information Systems," *Computer* 25, no. 3 (1992): 38-49, http://dx.doi.org/10.1109/2/121508.

13. Enrico Coiera, Martin Walther, Ken Nguyen, and Nigel H. Lovell, "Architecture for Knowledge-Based and Federated Search of Online Clinical Evidence," *Journal of Medical Internet Research* 7, no. 5 (2005), http://www.jmir.org/2005/5/e52/.

14. Shirley Cousins and Ashley Sanders, "Incorporating a Virtual Union Catalogue into the Wider Information Environment through the Application of Middleware: Interoperability Issues in Cross-database Access," *Journal of Documentation* 62, no. 1 (2006): 120-144, http://dx.doi.org/10.1108/00220410610642084.

15. "SRU Software and Tools," http://www.loc.gov/standards/sru/resources/tools.html; "Z39.50 Registry of Implementators," http://www.loc.gov/z3950/agency/register/entries.html.

16. "JAFER ToolKit Project," http://www.jafer.org.

17. "CQL-Java: a free CQL compiler for Java," http://zing/z3950.org/cql/java/.

18. Bojana Dimić, Branko Milosavljević and Dušan Surla,"XML Schema for UNIMARC and MARC 21 formats," *The Electronic Library* 28, no. 2 (2010): 245-262, http://dx.doi.org/10.1108/02640471011033611.

19. "UNIMARC formats and related documentation," http://www.ifla.org/en/publications/unimarc-formats-and-related-documentation.

20. "MARC 21 Format for Bibliographic Data," http://www.loc.gov/marc/bibliographic/.

21. "UNIMARCSlim XML Schema," http://www.bncf.firenze.sbn.it/progetti/unimarc/slim/documentation/unimarcslim.xsd.

22. "Marc21Slim XML Schema," http://www.loc.gov/standards/marcxml/schema/MARC21slim.xsd.

23. "DublinCore XML Schema," http://www.loc.gov/standards/sru/resources/dc-schema.xsd.

24. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software* (Indianapolis: Addison–Wesley, 1994), 315-323.

25. *Ibid.*

26. Danijela Boberić and Branko Milosavljević, "*Generating Library Material Reports in Software System BISIS,*" (Proceedings of the 4th international conference on engineering technologies - ICET, Novi Sad, 2009); Danijela Boberić and Dušan Surla, "XML Editor for Search and Retrieval of Bibliographic Records in the Z39.50 Standard", *The Electronic Library* 27, no. 3 (2009): 474-495, http://dx.doi.org/10.1108/02640470910966916 (accessed February 22, 1013); Bojana Dimić and Dušan Surla, "XML Editor for UNIMARC and MARC21 cataloguing," *The Electronic Library* 27, no. 3 (2009): 509-528, http://dx.doi.org/10.1108/02640470910966934 (accessed February 22, 2013); Jelena Rađenović, Branko Milosavljević and Dušan Surla, "Modelling and Implementation of Catalogue Cards using FreeMarker," *Program: electronic library and information systems* 43, no. 1 (2009): 63-76, http://dx.doi.org/10.1108/00330330934110 (accessed February 22, 2013); Danijela Tešendić, Branko Milosavljević and Dušan Surla, "A Library Circulation System for City and Special Libraries", *The Electronic Library* 27, no. 1 (2009): 162-186, http://dx.doi.org/10.1108/02640470910934669.

27. "Hessian," http://hessian.caucho.com/doc/hessian-overview.xtp.

28. Branko Milosavljević, Danijela Boberić, and Dušan Surla, "Retrieval of Bibliographic Records Using Apache Lucene," *The Electronic Library* 28, no. 4 (2010): 525-539, http://dx.doi.org/10.1108/02640471011065355.

ACKNOWLEDGEMENT

The work is partially supported by the Ministry of Education and Science of the Republic of Serbia, through project no. 174023: "*Intelligent techniques and their integration into wide-spectrum decision support.*"