Querying of Open Source Programs Libraries: an Approach Based on a XML Metadata Repository

Konan Marcellin BROU
Institut National Polytechnique F.H. Boigny, Department of Mathematics & Computer Science P.O. Box 1093 Yamoussoukro (Côte d'Ivoire)
Tel.: Fax: (225) 30640541 / 30640406, Email: konanmarcellin@yahoo.fr

Abstract

The conception of components that libraries of open source program contain, allows the reduction of production and maintenance cost of software. In order to use these libraries, the developer consults each library manually, and reads the documentation before choosing a component. Therefore, this process must be automated, mainly if several libraries are to be consulted. The semantic and significant aspects of each library and its components are displayed in an XML document called info-component, which is based on an UML model of metadata. This model has allowed us to implement an info-component repository on the libraries and their components, as well as methods and search techniques of components, which are based upon the XQuery request language. The user can have access to this repository via an Intranet network or Internet. A prototype has been designed and tested on some libraries, which include Lapack++, MV++ and SparseLib++ in the field of software computing.

1. Introduction

Current software applications have become more and more complex and expensive. Component reuse approach is a technique, which permits the development of new software by using components already existing rather than developing each system from the very beginning. The reuse of components must lead to more rapid development time and to lower costs. Therefore, it can improve productivity and the quality of the new components [1]. Although there exist many library programs whose access is free (open-source for instance), these components are not always well organized and documented. As a result, choosing a component becomes long and complex. The search process is one of the critical points of reuse because a developer does not necessarily know the existence of components that he/she wish to create.

This work aims at improving the process of searching, localizing and extracting component software in the context of PSRep (Pau Software Repository), which is a managing system of source code library developed at the University of Pau for the reuse of software. The PSRep project was initiated at a search convention for the development of a catalog and software component browser required by the IFP (Institut Français du Pétrole) [2] in the field of software computing.

In order to use these libraries, the developer manually consults each library, reads the documentation before choosing component software. This task aims at automating the reuse process of these components, especially if there is a need to consult several libraries. A component is software, which does a favor by means of an external interface [3]. As far as we are concerned, a component is a class or a function. There exist two kinds of components: the "black box" components whose access is exclusively via their interface, and the "white box" components whose source code is accessible. We are interested in the reuse of "white box" components in order to reduce the production cost and maintenance of the software. In order to make reuse increase productivity in software development [4], three basic operations are essentially carried out. These operations are: (i) cataloging the components, (ii) searching the components, (iii) re-using components as they have been found or adapting them to their needs.

Finding an adequate component for a given application goes either via a "linear" consultation of diverse components of the platform (services), or through a request process based criteria, which can appear very numerous and complex. To take advantage of these components, the management system of the library programs must be completed by a semantic description of the components, which have been extracted from their code; we call them "internal metadata". They must also be completed by an index description (author, creation date, language etc…); they are independent from data that we call “external metadata”. These metadata respect the requirements of the metadata of the Dublin Core [5], which is referred to in the field of electronic resource description.

We conceptualize metadata as a bunch of descriptors, which can identify specific components. The semantic and index aspects of the descriptors are integrated into an UML model of metadata. This model has allowed us to implement a metadata base on the libraries and their components. Each library and its components are described according to this model, and are represented in a XML document that we call (info-component), and which respects a DTD as a translator of this UML model. The metadata base (or catalog) is composed of as many info-components as there are libraries. The XML document base is then an info-component repository. This model has also allowed us to implement methods and techniques of component search via info-component repository and language request XQuery [6].

An interface has been conceived in order to automate or provide some help for the reuse of these components. This help allows: (1) the automatic extraction of the internal metadata from a given library; (2) the manual indexation of the external metadata from a form or an XML document in which these specific information have been stocked; (3) the updating of the info-component repository; (4) the consultation and surfing in the repository; (5) the search of components from the repository info-component (use of master request XQuery); (6) the extraction of a component from the base of program library.

The interest and the contribution of this work can be summarized as follows: (1) Creation of model for the structure of components within an XML document repository; this assures some inter operation with other tools of development of component-based software; (2) Use of metadata and thesaurus, which favor search via classic search channels; (3) Generic solution that can be applied to any kind of component library, whose source code is accessible.

In this paper, we specifically focalize our interest on the proposed metadata UML model and its implementation as an XML database document, as well as the development of question tools with the help of XQuery request. A prototype has been developed and tested on some libraries like Lapack++ [7], MV++ [8] and SparseLib ++ [9] in the field of software computing.

This article is organized as follows: in section 2, we present some work on component reuse. Then in section 3, we detail our approach which rests upon a metadata base on components and on an info-component repository. Later in the section 4, we present the component search process. Eventually in section 5, we present the prototype implementation process, which allows to stock components and do search on them, together with a conclusion that comes after.

2. Software component management

The management of components includes two aspects: the creation of model for components in order to stock them, and the query process to retrieve them. Amongst the diverse undertaken works, several are about component search and reuse. As far as [1] is concerned, it is simply to reuse existing components in any software developmental phase. Its architecture uses a mediation layer, which integrates the web semantic component with the acknowledged components. They are recorded in virtual library components. These components are described through XML and published by a local repository or distant server. They use field ontology for the reuse of components. On its part, [10] develops strategies or tools to assist the developer in the process of testing and assessing component candidates. It uses an XML schema for the description of components. Works in [11] are based on efficient component search via key words or out of similarity between components. [12] proposes a component search tool called “black box”, which uses a field ontology linked to a repository. It proposes a component description model and different methods of search based on the text, the lexis or a formal specification.

These works just interest “black box” software components, which are exclusively accessible through their interfaces. Therefore, it is impossible to have access to their source code. As a result, reusing process takes place on the whole component, and the problem that needs to be solved simply concerns the selection of the required component. Our task then consists in finding and re-using not only part of the source code of the components, but also in carrying out delicate search, which is based on keywords contained in their source code or their documentation. There exist different approaches, which permit to apprehend the problem of the reuse of the components [13]: to develop in order to be used, and to use in order to develop. We side with the second approach, which consists on the one hand in exploiting the component source code, and on the other hand, in exploiting the external documentation, which goes with the library, if it exists. Such documentation software as Doxygen and JavaDoc automatically extracts the internal metadata, which are embedded in the component source code, in order to feed an XML document base.

3. Software component metadata

A catalog is defined as an information base, which describes functional and structural properties of several program libraries. This information is metadata represented in documents that are called info-components. Metadata or data on a datum, give information on the nature and characteristics of other data. They help users in their attempt to discover the existence of resources and the nature of what they look for [14]. They are used to look for, to reuse, to disseminate, to publish multiple contents (texts, images, video, etc.). Our catalog is metadata (commonly called index), which describe info-components made of two sorts of descriptors: index descriptors and semantic descriptors.

3.1. Index descriptors

Index descriptors, which are called "external metadata", exclusively allow the description of program library without reference to its content (i.e. its components). These descriptors are manually extracted from the external documentation that goes with them, and which are stored in an XML document: name, language, version, field, etc… These metadata fit in the Dublin Core [5] metadata norm so that a reference can easily be given to the info-components through classic search channels. This norm is a bunch of 15 metadata elements relating to: **Content**: Title, Description, Subject, Coverage, Type, Relation; **Intellectual property**: Creator, Contributor, Publisher, Rights; and **Version**: Date, Format, Identify, and Language.

3.2. Semantic descriptors

The semantic descriptors, which are commonly called "internal metadata", allow the exclusive description of component functional proprieties. They are automatically extracted from component source code. Firstly, the principle consists in producing documents in which is integrated the semantic description of components, by using such document software as Doxygen and JavaDoc. Secondly, these documents are analyzed in order to extract such descriptors as the name of the components, their source files, the method name and their signatures etc… (Figure 2.). The metadata conceptual method for an info-component that we present in next paragraph allows to make a model of these two types of descriptors.

3.3. Metadata model for Info-component

After studying the running and analyzing the yielded documents via documentation software, the data model for an info-component as formally translated in UML (Figure 1.). This model includes two parts: one part allows the index description of components: Library, Documentation, Author, Platform, Keywords, and another part describes the semantic description of components, which have been automatically extracted from their code source (Component, FileSrc, Service).

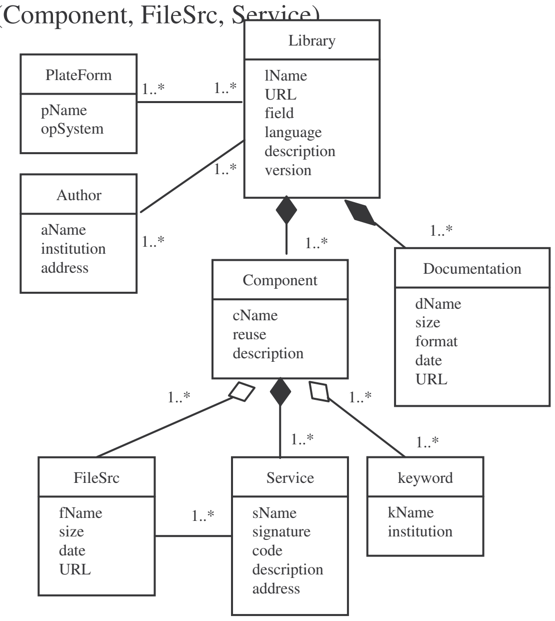


Figure 1. Metadata model for info-component.

3.4. XML document model for an Info-component

XML (eXtensible Markup Language) allows the representation of structured and non structured data; it therefore facilitates the automated processing of documents and data. An XML document possesses a labeled tree-structure. The rules, which permit the rigorous building of such a tree, are provided by a DTD (Document Type Definition). Our XML document model for an info-component is based on a DTD, which derives from the UML metadata model as described below (Figure 1.). This transformation can be realized by such a tool as ArgoUML [22].

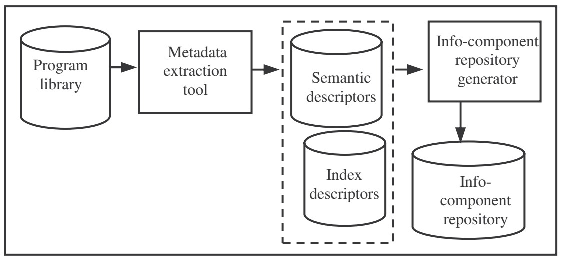


Figure 2. Process of generating info-component repository

Each library program is represented by a well formed and valid XML document (info-component) which respects this DTD. An info-component is an XML document which contains the index and semantic description of components. The info-component repository is constituted by the whole bunch of these XML documents.

4. Component search

There exist two kinds of search: surfing search and questioning search. Surfing search consists in looking for a component via “idea association” by scrolling the info-components as it occurs in a hypertext. As far as the questioning search that we present in forthcoming paragraphs, it consists in submitting requests to the info-component repository. These requests are based on the use of keywords coming from a thesaurus.

4.1. XML thesaurus model

A thesaurus is defined as a tool which traces lexis which is specific to a given field. It includes a number of terms (descriptors and non-descriptors) and relations, which specify their semantic environment [15] [16]. Our thesaurus turns the field concepts into models with the help of terminology information, which allows a structural but informal representation of knowledge. The structure of the concept contains linguistic knowledge which relate to its equivalent terms, and conceptual knowledge, which put it into a hierarchy. The notion of terminological file is used to describe the concepts thanks to a common structure. Each field of this concept is called “terminological information” (term, language, definition, generic concepts, sub-concepts, synonym, homonym, linked concepts, and image) [17].

Two types of links are turned into models: (1) *vertical links* which represent hierarchical relations existing between concepts (generalization, specification). Notions of generic concepts and sub-concepts are re-found there. (2) Horizontal links which correspond to reference links, i.e. the semantic relations existing between the concepts. These links represent the notions of synonymy, homonymy and linked concepts. Horizontal links are used to trace component via similarity process. This approach is interesting since not only can it permit to find the correct component, but also all those which are linked by any conceptual relation [18]. The following DTD allows people to turn the thesaurus into model.

```
<!ELEMENT domain (concept+)>
<!ELEMENT concept (term, language, image, definition,
context, rc*,keyword+)>
<!ATTLIST concept id CDATA #REQUIRED>
<!ELEMENT term (#PCDATA)>
<!ELEMENT language (#PCDATA)>
<!ELEMENT image (#PCDATA)>
<!ELEMENT definition (#PCDATA)>
<!ELEMENT context (#PCDATA)>
<!ELEMENT rc (cg*, cl*, sc*,synonym*, homonym*)>
<!ELEMENT cg (reference*)>
<!ELEMENT sc (reference*)>
<!ELEMENT cl (reference*)>
<!ELEMENT synonym (reference*)>
<!ELEMENT homonym (reference*)>
<!ELEMENT reference (#PCDATA)>
<!ATTLIST reference lien CDATA #REQUIRED>
<!ELEMENT keyword (name)>
<!ATTLIST keyword bib CDATA "">
<!ELEMENT name (#PCDATA)>
```

All the relations (hierarchical and semantic) which exist between the concepts are represented by the attribute *link* of the *reference* element. This allows all the links to be managed in the same way.

4.2. Search process

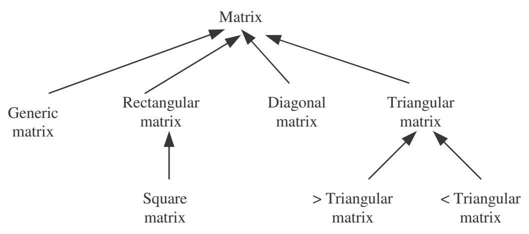


Figure 3. Domain thesaurus.

Through the thesaurus field, users can look for components in a transparent and uniform way, without referring to their location. The process of looking for components occurs as follows: the user submits her/his request via the simple use of terms in the thesaurus (Figure 3.); the request Manager determines the names of the components, which correspond these terms before translating the user request into XQuery requests. For the time being the thesaurus is built manually as follows:

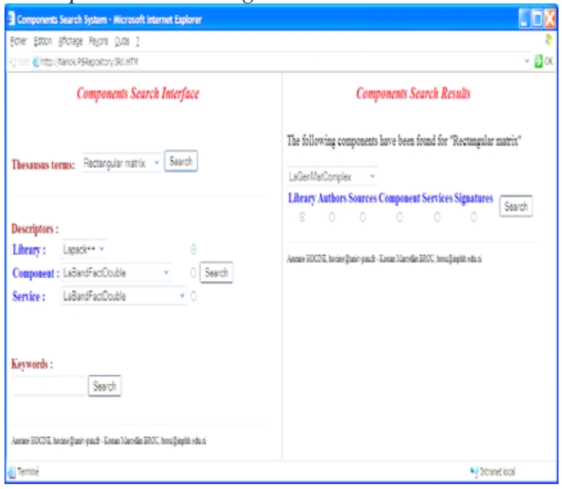
```
<domain>
  <concept id="2">
   <term>Rectangular matrix</term>
   <language>English</language>
   <definition>Matrix having number of lines and number
  of columns different</definition>
   <rc><cg><reference lien="1"/></cg>
    <cl><reference lien="3"/><reference lien="4"/>
  </cl>
   </rc>
  <keyword bib="Lapack++">LaGenMatDouble
  </keyword>
  <keyword bib="Lapack++">LaGenMatComplex
  </keyword>
  <keyword bib="MV++"> MV_Mat_Int </keyword>
  </concept>
</domain>
```

4.2.1. XQuery Request: XML Query language or XQuery is a language for XML, and it is defined by W3C for the different types of XML applications; it is a statement language, and remains similar to SQL language. However, SQL language manipulates tables (integer n-uplets) whereas XQuery manipulates tree sequences. XQuery is a deeply typical functional language; it supports classic task processing and questioning XML applications. It contains Xpath expressions in order to scroll and extract XML fragments of documents, and expressions to build new XML documents. The expression flower: FWLR (For-Let-Where-Return) permits to build, trees from another tree. Its syntax is as follows: ***for a in f where c return a'*** with:

- ***for***: permits to have access to another **a** of forest **f**
- ***where***: permits to check if tree **a** answers condition **c**
- ***return***: permits to build a new tree **a'**, from tree **a**

The ***let*** clause of the syntax let ***$var: = exp*** affects one phrase to another.

4.2.2. Request form: A request form as in Figure 4. enables people to submit requests in a graphic form without knowing the structure and the storage place of the info-component repository. Here is an example of request: "*Searching for components which process "rectangular matrix*".



Components Search System - Microsoft Internet Explorer
Fichier Edition Affichage Favoris Outils ?
http://Nanok/PSRepository/SRI.HTM
OK
Search
Terminé
Intranet local

Components Search Interface

Thesaurus terms: Rectangular matrix

Descriptors :
Library : Lapack++
Component : LaBandFactDouble
Service : LaBandFactDouble

Keywords :

Amine HOCINE, hocine@univ-pau.fr - Konan Marcellin BROU, brou@inphb.edu.ci

Components Search Results

The following components have been found for "Rectangular matrix"

LaGenMatComplex

Library Authors Sources Component Services Signatures

Amine HOCINE, hocine@univ-pau.fr - Konan Marcellin BROU, brou@inphb.edu.ci

Figure 4. Request form.

After selecting the term of the thesaurus, "*rectangular matrix*", the system finds the names of corresponding components: LaGenMatDouble, LaGenMatComplex… The request can then be refined in the second part of the interface by choosing a component name, and an index or semantic descriptor. The graphic request is then translated into XQuery request.

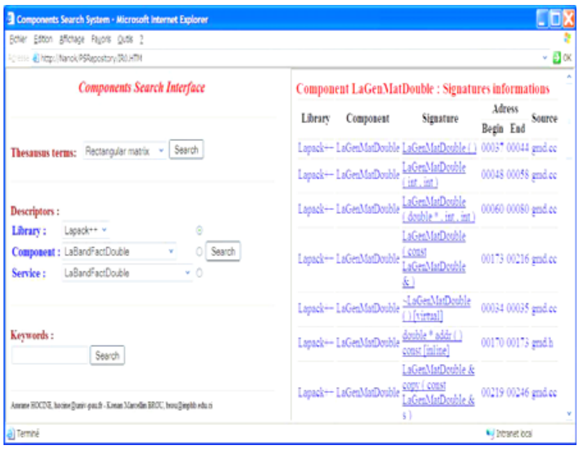
4.2.3. Translation into XQuery: The graphic request is then translated into XQuery request through the use of request "models" or parameter request. The request "model" permits to find the signatures of component methods which are linked to a thesaurus term in the following form:

```
for $t in document("thesaurus.xml")//domain/concept
where $t/term=$term return
 for $m in $t/keyword return
  if ($m/text() = $cName) then
   let $b : = concact($m/@bib, "XML.xm1")
   let $c :=$m/text() return
   let $x : document($b)//library
   for $y in document($b)//library/component
   where $y/cName = $c return
   <signatures>{$y/service/signature}</signatures>
  else
   ()
```

The request "model" contains free variables $term, $cName and $b, which will be respectively instancied by a term form the thesaurus, a name of component and an info-component. The request, which permits to find the signatures of a component method LaGenMatDouble linked to the thesaurus' rectangular matrix, is translated into XQuery as follows:

```
for $t in document("thesaurus.xml")//domain/concept
where $t/term="Rectangular matrix" return
 for $m in $t/keyword return
  if ($m/text() = "LaGenMatDouble") then
   let $b:= concact($m/@bib, "XML.xml")
   let $c = $m/text() return
   let $x:= document ($b)//library
   for $y in document ($b)//library/component
   where $y/cName = $c return
    <signatures> {$y/service/signature}</signatures>
  else
   ()
```

4.2.4 Conceiving the answer and presenting the result:



Components Search System - Microsoft Internet Explorer
Fichier Edition Affichage Favoris Outils ?
http://Nanok/PSRepository/SRI.HTM
OK
Search
Terminé
Intranet local

Components Search Interface

Thesaurus terms: Rectangular matrix

Descriptors :
Library : Lapack++
Component : LaBandFactDouble
Service : LaBandFactDouble

Keywords :

Amine HOCINE, hocine@univ-pau.fr - Konan Marcellin BROU, brou@inphb.edu.ci

Component LaGenMatDouble : Signatures informations

Library	Component	Signature	Adress Begin	Adress End	Source
Lapack++	LaGenMatDouble	LaGenMatDouble ( )	00037	00044	gmd.cc
Lapack++	LaGenMatDouble	LaGenMatDouble (int , int)	00048	00058	gmd.cc
Lapack++	LaGenMatDouble	LaGenMatDouble (double * , int , int)	00060	00080	gmd.cc
Lapack++	LaGenMatDouble	LaGenMatDouble (const LaGenMatDouble &)	00173	00216	gmd.cc
Lapack++	LaGenMatDouble	~LaGenMatDouble ( ) [virtual]	00034	00035	gmd.cc
Lapack++	LaGenMatDouble	double * addr ( ) const [inline]	00170	00173	gmd.h
Lapack++	LaGenMatDouble	LaGenMatDouble & copy (const LaGenMatDouble & s)	00219	00246	gmd.cc

Figure 5. Request execution result.

The request manager instancies the free variables of the request "model" before transmitting the request to the XQuery engine for its implementation. The result of the implementation is an XML document, which is polished with XLS and CSS style sheet of paper before presenting the final result to the user (See Figure 5.).

5. Implementation of the prototype

The documentation software that we have used is Doxygen [19]. It is an open source software which permits to generate the documentation on the components, starting from the source code of a program C, C++, Java, and Objective C. Moreover, if the comments are written in a precise syntax, Doxygen will capture them in the documentation. Doxygen provides the following information from the sources: list of functions, dependence graph, (what functions call for which?), list of data structures, list of classes and hierarchy. The documentation can be generated through several formats: HTML, LaTeX, RTF, PostScript, and PDF. For the time being, we have decided to generate HTML documentation from C++ programs. As a matter of fact, PSRep possesses some tools which allow transforming HTML documents into formats which are easier to analyze in order to extract the metadata on the components.

5.1. The architecture of the prototype

The Figure 6. displays the system which permits the management and the exploitation of the info-component repository. It includes two parts: one part allows the management of the info-component repository, and another part deals with the search of components.

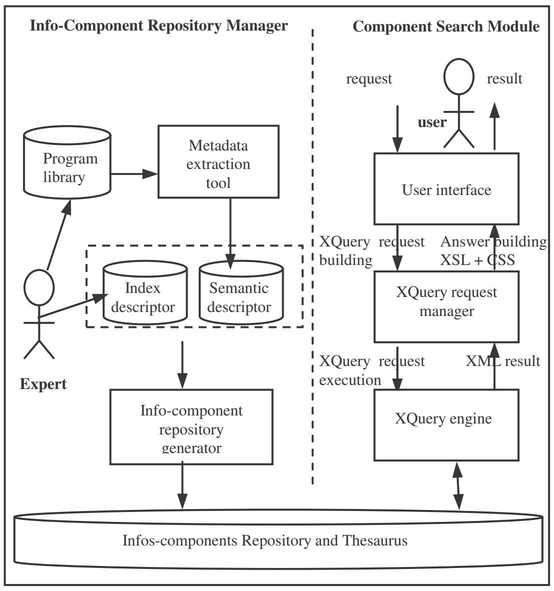


Figure 6. The architecture of the system.

5.1.1. Info-component repository manager: It provides services for the analysis, extraction and storage of metadata about components. It also allows the creation of info-components.

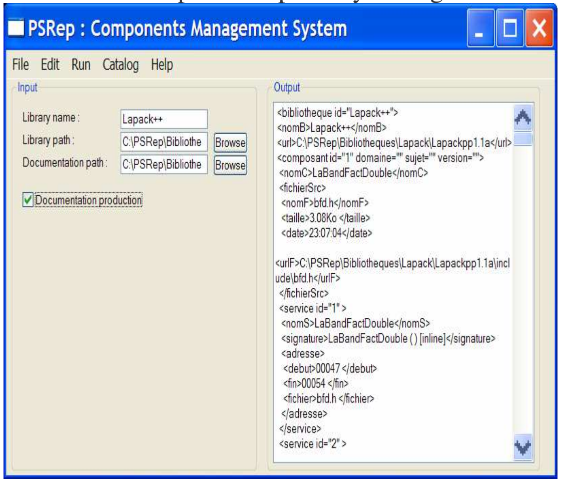
- Extraction tool: It allows the semantic description of a library and its components.
- Info-component repository generator: It allows the automatic generation of an info-component from the index and semantic description of a library and its components.

5.1.2. Search module: It is in charge of the building of requests, their implementation and the presentation of the result to the user.

- User interface: it allows the computer to create an XQuery request without knowing the XML structure of the catalog. Its functions are: to receive a request based on the terms of thesaurus field; to have the result of the request into XML of the XQuery request manager, to convert it into HTML with XSL and CSS paper style, and display it on the screen.
- XQuery request manager: Its role is to: (1) translate a user request from its corresponding initial form into XQuery before going to the XQuery Engine; (2) receive the answer of Engine XQuery request; (3) transform the results into XML data; (4) Return the result in XML to the interface user.
- XQuery engine: Its role is to carry out the XQuery request and to deliver the XML result to the XQuery request manager.

5.2. The interface of the prototype

The interface of the system, as in (Figure 7.), is used by the administrator for the component catalog and the info-component repository management.



Library name : Lapack++
Library path : C:\PSRep\Bibliothe
Documentation path : C:\PSRep\Bibliothe
[✓ Documentation production]

```
<bibliotheque id="Lapack++">
<nomB>Lapack++</nomB>
<url>C:\PSRep\Bibliotheques\Lapack\Lapackpp1.1a</url>
<composant id="1" domaine="" sujet="" version="">
 <nomC>LaBandFactDouble</nomC>
 <fichierSrc>
  <nomF>bfd.h</nomF>
  <taille>3.08Ko </taille>
  <date>23:07:04</date>

<urlF>C:\PSRep\Bibliotheques\Lapack\Lapackpp1.1a\incl
ude\bfd.h</urlF>
 </fichierSrc>
 <service id="1" >
  <nomS>LaBandFactDouble</nomS>
  <signature>LaBandFactDouble ( ) [inline]</signature>
  <adresse>
   <debut>00047 </debut>
   <fin>00054 </fin>
   <fichier>bfd.h </fichier>
  </adresse>
 </service>
 <service id="2" >
```

Figure 7. Catalog system interface.

The development environment is essentially based upon language Python [21] (for the extraction and the storage of metadata on the components), Galax [20]: an open source implementation of the language XQuery (for information search), language PHP and Apache web server (for the creation and management of website). The prototype that we have developed has been tested with the following libraries: Lapack++, MV++ and SparseLib++ (see Table 1).

Table 1. Library number of components and services.

Library	Number of components	Number of services
Lapack++	36	753
MV++	14	310
SparseLib++	22	440

6. Conclusion

All along this paper, our objective was to show how to calculate free components for reuse purpose. To accomplish this task, we have proposed a system of component catalog. This system is based on the extraction of metadata which are contained in the documentation on components. This documentation has been generated by documentation software. These metadata are stored in an info-component repository. The search of component is performed by a friendly interface conceived around XQuery language. One prototype of the component cataloguing has been developed and successfully tested on MV++, Lapack++ and SparseLib++. Forthcoming works will be about the improvement of the search process by integrating an engine search which permits the use of some keywords. It is question of providing the users advanced tools which are founded on XQuery request language, allowing them to express request naturally in order to get semantically pertinent answers.

6. References

[1] R.P. de Souza, M.N. Costa, and R.M.M. Braga, "Software Components Reuse Through Web Search and Retrieval", Journal of the Brazilian Computer Society, vol.8 no.2, Campinas, Nov. 2002, pp 55-63.

[2] A. Hocine, P. Raffinat 200, Etude et prototypage d'un système interactif de catalogage de composants, Rapport Final, Convention IFP-UPPA, Décembre 2002.

[3] Projet ACCORD, "Assemblage de Composants par Contrats en environnement Ouvert et Réparti", juin 2002, http://www.infres.enst.fr/projets/accord/lot1/lot_1.1-1.pdf.

[4] M. C. Lafaye, G. Louis, D. Mehira, "Système de Recherche de Composants Logiciels, Application à la Réutilisation en Synthèse d'images", Congres INFORSID'97, Toulouse, 10-13 juin 1997, pp. 601-621.

[5] "Dublin Core Metadata Initiative", http:// dublincore.org/documents/.

[6] W3C XML, "Query, XQuery 1.0 and XPath 2.0 Full-Text", http://www.w3.org/TR/2004/WD-xquery-full-text-20040709/.

[7] Lapack++, "Linear Algebra PACKage in C++", http://math.nist.gov/lapack++/.

[8]: MV++, "Numerical Matrix/Vector Classes in C++", http://math.nist.gov/mv++/.

[9] SparseLib++,"A Sparse Matrix Library in C++ for High Performance Architectures", http://math.nist.gov/ sparselib++/.

[10] V. Maxville, J. Armarego, and C. Peng Lam. "Intelligent Component Selection," Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04), vol. 01 no. 1, pp. 244-249, 28th 2004.

[11] A. Michail, and D. Notkin, "Assessing Software Libraries by Browsing Similar Classes, Functions, and Relationships", Proceedings of the 21st international conference on Software engineering Los Angeles, California, United States, 1999, pp. 463 – 472.

[12] R. Meling, E.J. Montgomery, "Storing and Retrieving Software Comonents: A Component Description Manager", Proceedings of ASWEC'2000 -The Australian Software Engineering Conference, IEEE CS Press, Los Alamitos, California, April 28-30, pp. 107-117.

[13] O. Khayati, A. Conte, J.P. Giraudin, "Les systèmes de recherche de composants", Système de composants adaptables et extensible – 17 et 18 oct. Grenoble 2002, pp. 232-240 http://rangiroa.essi.fr/arcad/livrables/proceedings-JC2002.pdf.

[14] Patrick Peccatte, "Métadonnées: une initiation Dublin Core, IPTC, EXIF, RDF, XMP", http:// peccatte.karefil.com/software/Metadata.htm.

[15] AFNOR 1981, "Règles d'établissement des thésaurus monolingues", AFNOR NF Z47-100.

[16] E. BRUNO, J. LE MAITRE et E. MURISASCO, "Indexation et interrogation de photos de presse décrites en MPEG-7 et stockées dans une base de données XML", Ingénierie des systèmes d'information (RSTD - ISI), vol. 7, n° 5-6, 2002, pp. 169-186.

[17] K. M. BROU, "Le modèle de représentation et de Gestion Hypertextes des Concepts d'un Domaine dans le Système CoDB-Web", Revue Rint (Réseau international de néologie et de terminologie) N°19, 1999, pp. 89-100.

[18] D. Lucredio, A. F. do Prado, and E. S. de Almeida, "A Survey on Software Components Search and Retrieval" EUROMICRO, vol. 00, pp. 152-159, 30th 2004.

[19] Doxygen, "Documentation System for C++, C, Java, Objective-C, Python, IDL" http://www.stack.nl/~dimitri/ doxygen/.

[20] Galax, "An open-source implementation of Xquery 1.0" http://www.galaxquery.org/.

[21] Python, "An interpreted interactive object-oriented programming language" http://www.python.org/.

[22] ArgoUML, "A modeling tool that helps you do your design using UML" http://argouml.tigris.org.