International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:3, 2007

A Multilanguage Source Code Retrieval System Using Structural-Semantic Fingerprints

Mohamed Amine Ouddan, and Hassane Essafi

***Abstract*—**Source code retrieval is of immense importance in the software engineering field. The complex tasks of retrieving and extracting information from source code documents is vital in the development cycle of the large software systems. The two main subtasks which result from these activities are code duplication prevention and plagiarism detection. In this paper, we propose a multilanguage source code retrieval system based on two-level fingerprint representation, respectively the structural and the semantic information within a source code. A sequence alignment technique is applied on these fingerprints in order to quantify the similarity between source code portions. The specific purpose of the system is to detect plagiarism and duplicated code between programs written in different programming languages belonging to the same class, such as C, C++, Java and CSharp. These four languages are supported by the actual version of the system which is designed such that it may be easily adapted for any programming language.

***Keywords*—**Source code retrieval, plagiarism detection, clone detection, sequence alignment.

I. Introduction

IN a world dominated by the information technology the internal communication channels of any company are packed with sensitive, confidential information and digital assets such as R&D developments and software source codes. With the explosive amount of accessible information in the recent years, corporations are naturally apprehensive that sensitive information might find its way into the hands of competitors or even on the Internet network. Therefore it is essential that companies prevent the leak of its confidential information and protects its intellectual property rights. Software source codes are a part of this confidential information and provide vital support for the development and the prosperity of any company.

Two activities result from these requirements, the fist one consist of protecting the software against illicit exploitation and detecting different source code plagiarism cases. With the evolution of Internet and the search engines the free access to the source code makes plagiarism of the open-source software possible without respecting affiliated licenses.

The second activity consists of keeping the software system up-to-date and functioning properly [1][2][3]. The constant evolution of the software requires at the same time continuous modifications and maintenance of the source codes. Duplicated code complicates this activity and leads to higher maintenance cost because the same bugs will need to be fixed and consequently more code will need to be tested. It is reported by Burd[4] that “during the maintenance of legacy code, it is common to identify areas of replicated code”, and it is suspected that quantity of the duplicated code is in general between 5% to 10% and can go up to 50% [5][6][7].

In this paper, we propose a multilanguage source code retrieval system using an original similarity measure approach, which is based on the characterization of the source code by extracting and combining structural and semantic information. Using the concept of Grammar-Actions a set of salient components and salient elements are detected and characterized by sequences called respectively "Structural sequences" and "Genetic sequences". The sequences are composed of specific symbols which constitute the knowledge resources of each programming language (C, C++, Java and CSharp). These resources reflect the main structural and semantic features of each language. As shown in Fig. 1, the sequences are embedded into two-level fingerprint structure, where the first level encloses the structural content of the source code and the second level contains the semantic one. Applying sequences alignment technique, we quantify the similarity between two fingerprints which is considered as an abstraction of plagiarism rate (or duplicated rate). Fig. 1 illustrates the architecture of the system. A detection language phase is applied to each code in order to use the adequate parser and to load the corresponding knowledge resources.

The paper is structured as follows: In section II, we present the main related works. The fingerprint extraction and the similarity measure are respectively described in sections III and IV. In section V, we present a set of experiment results in order to evaluate the robustness of the system against several source code transformations. Finally, we present our conclusions and perspectives in section VI.

Manuscript received May 25, 2007.

Mohamed Amine Ouddan is with Marne la Vallée University (ICMS) and R&D department, Advestigo compagny, 140 Bureaux de la Collines 92210 Saint-Cloud, France (phone: +33 1 72 77 70 13; fax: +33 1 46 89 68 60; e-mail: amine.ouddan@advestigo.com).

Hassane Essafi is with R&D department, Advestigo compagny, 140 Bureaux de la Collines 92210 Saint-Cloud, France (phone: +33 1 72 77 70 03; fax: +33 1 46 89 68 60; e-mail: hassane.essafi@advestigo.com).

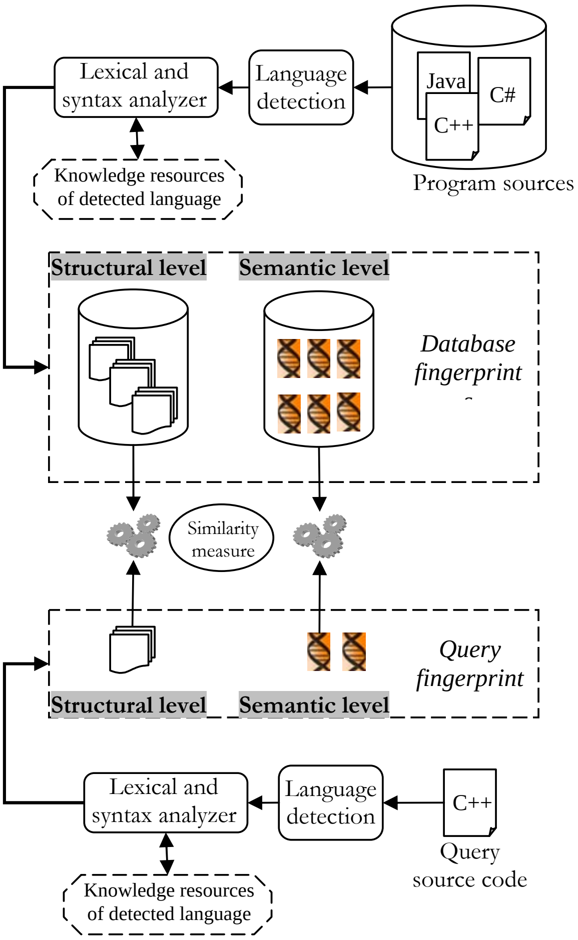


Fig. 1 Framework of the multilanguage source code retrieval system

II. Related Works

Similarity measure between two source codes relies on their representation. Different approaches have been proposed towards the source code content access, which can be classified into two main categories according to their representation model. These categories regroup respectively the statistical approaches and the structural approaches.

A. Statistical Approaches

Statistical approaches are based on the vector-space model where each source code is represented by a characteristic vector, and the similarity between codes is then calculated by a distance measure in this space. Using the Halstead's software science metrics [8], the characteristic vector is computed in order to reflect the complexity of the represented source code. In [9] a vector of four metrics is assigned to the source code representing respectively the number of operator occurrences, the number of operand occurrences, the number of distinct operators, and the number of the distinct operands.

To improve the robustness of the characteristic vector and the precision of the similarity measure, several metrics were proposed [10] such as the number of variables, functions, conditional statements, iterative statements, and assignment statements. Faidhi and Robinson [11] have nominate a set of uncorrelated metrics like the average number of characters per line, the conditional and iterative statement percentage, the average identifier length, etc.

B. Structural Approaches

The structural approaches are more powerful than the statistical ones for the plagiarism detection [12] due to the poor representation of the vector-space model in terms of structural information. Hence, the aim idea of the structural approaches is to convert the source code content into a compact representation according a specific model in order to encode the basic and relevant structural information. The two main models used in this way are the conceptual graphs model and token strings model.

The conceptual graph model proposed by John Sowa [13] and used in the knowledge representation area was applied in the context of source code retrieval [14]. Each node reflect either a structural concept (such as statements, function, variable, etc) or structural relation between two concepts (such as condition for branching statement, a concept is a parameter of another concept, dependence between two concepts, etc).

In the second model the approach consist of converting the source code into a sequence of tokens and then using an appropriate sequence comparison algorithm to find similar tokens. The most important tools in the context of plagiarism detection, which use the token string representation, are YAP3[15] Jpalg[16] and Moss[17].

III. Source Code Fingerprint Extraction

In order to extract the main features from each source code and construct its fingerprint, structural and semantic content access is based on the grammar of the programming language denoted G_L. Thus the grammar G_L must be synchronized with a set of actions called "characteristic actions" which allow the translation of the source code from the programming language to the fingerprint language. Each programming language has its own syntax according a set of grammatical specifications, which reflect the structural and the semantic features of the corresponding source codes. Therefore, the fingerprint extraction must take into account both the structural and the semantic content of the source code in order to improve the pertinence and the precision of the retrieval task.

A. The Concept of Grammar-Actions

The notion of trace is an universal characteristic for many programming languages which is defined by Hoare [18] as an association between the program and the result which its provides: "It should be possible to associate with each program of a language a set of possible traces of the execution of that program; this association provides a 'mechanism' formal definition of the language ". Thus, in the context of characterization we distinguish two kinds of traces that result during the parsing of the source code. Syntactic trace which reflect the structural features of the parsed code, and execution trace which reflect its semantic features.

As mentioned previously, we have to construct a translator that allow conversion of structural and semantic contents into a two-level fingerprint, which is considered respectively as an abstraction of the syntactic and the semantic traces. The translator is based on the concept of Grammar-Actions where the aim idea is to assign significance (in context of characterization and traceability) to the parsing process. Therefore, with each grammar rule we associate a set of actions to be executed each time the grammar rule is recognized during the parsing process. These actions may achieve four vital tasks for our approach:

1. Extraction of salient components: the modularity feature supports partial plagiarism. Indeed it is important to protect both the totality of the code and its independent portions. A salient component is defined as a portion of the code that can be re-used separately in another code in different context (e.g. the plagiarized code). The concept of salient component differs according to the programming language, thus the task of extraction must be harmonized with the grammar of the language.

2. Generate structural sequences: this task reflect the notion of modular characterization. Therefore, for each salient component a structural sequence is generated in order to characterize both the totality of the code and all its independent portions.

3. Extraction of salient elements: the goal idea of this task is to detect the most elementary entities within a source code, i.e. the dynamic elements which achieve the interaction between the different portions of the code where their modification (or deletion) affect the behavior of the related program. The concept of salient element varies according to the programming language. As in the first task, the extraction of salient elements is based on the grammar of the language.

4. Generate genetic sequences: this task is considered to be an abstraction of the semantic trace. Consequently, for each salient element a genetic sequence is generated in order to characterize its activity and traceability during the program execution.

Required knowledge resources specific to the programming language must be constructed in order to achieve the tasks 2 and 4 (generation of the structural and semantic sequences).

The "Grammar Dictionary" [19] represent the first kind of resource which is denoted *GD$_L$* and considered as an association between the main structural programming concepts within the language *L* and the corresponding structural symbols. Formally, the *GD$_L$* is defined as a binary relation from the programming concept set to the structural symbols set:

$$GD_L : Concepts \rightarrow StructuralSymbols$$
$$R_i \quad \rightarrow \quad a_j$$

The "Semantic Dictionary" represent the second kind of resource which is denoted *SD$_L$* and considered as an association between all the operations that describe the activity of the salient elements and the corresponding semantic symbols. Formally, the *SD$_L$* is defined as a binary relation between the allowed operations set to the semantic symbols set:

$$SD_L : Operations \rightarrow SemanticSymbols$$
$$op_i \quad \rightarrow \quad b_j$$

B. Structural-Level Fingerprint

The structural-level fingerprint symbolizes an abstraction of the syntactic trace, which reflects the main structural features of the source code. Using the Grammar Dictionary each salient component is characterized by a sequence of structural symbols which are referred to the programming concepts that were recognized during the parsing process. The structural sequence of the salient component *A$_i$* is defined as:

$$S_{Ai} =< a_1, a_2, ..., a_n > \;/\; a_j \in GD_L, \forall j = 1,...,n$$

The structural symbols must characterize the essence content of the source code, i.e. all the main programming concepts within the language L

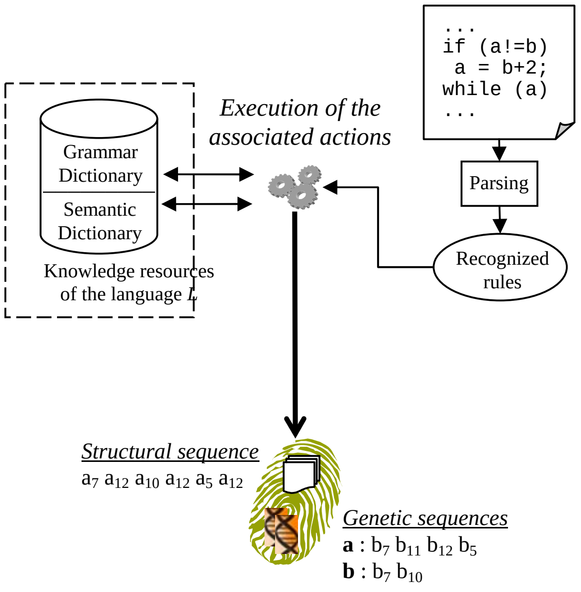
C. Semantic-Level Fingerprint

The semantic-level fingerprint describes the activity and the traceability of the salient elements throughout the program execution. This traceability is represented by the succession of the operations where the salient element participates such as arithmetic expressions or evaluation of a "while" loop conditions. Therefore, using the Semantic Dictionary each salient element is characterized by a sequence of semantic symbols which is considered as a signal reflecting the evolution of the element during the execution. The genetic sequence of the salient element *Bi* is defined bellow:

$$S_{Bi} =< b_1, b_2, ..., b_m > \;/\; b_j \in SD_L, \forall j = 1,...,m$$

Thus, the genetic sequences of each salient element within the source code characterize its semantic content. The semantic symbols must characterize all the main operations allowed by the language *L* which represent the essence of its semantic.

Fig. 2 recapitulates the fingerprint extraction scheme of any source code written in the programming language *L*:



```
...
if (a!=b)
 a = b+2;
while (a)
...
```

Fig. 2 Fingerprint extraction

D. *Multilanguage Characterization*

A multilanguage characterization is possible due to the modular architecture of the Grammar-Actions concept. Therefore our system allows identical characterization of two similar codes written in different languages such as C/C++ and Java. This property requires homogenizing both the Structural Dictionary and the Semantic Dictionary, i.e. the programming concepts which are presents in the two languages must be characterized by the same symbol:

$$GD_{C/C++} \cap GD_{Java} \neq \{\ \} \text{ and } SD_{C/C++} \cap SD_{Java} \neq \{\ \}$$

For example the concept of iterative statement is present in the four languages C/C++, java and CSharp and must be characterized by the same structural and semantic symbol. This property allows more relevance against the translation of the programming language used in several plagiarism cases. It is also helpful during the maintenance of the multilanguage software systems.

IV. The Similarity Measure

As mentioned above, the similarity measure consists of quantifying the plagiarism rate (or duplicated rate) between codes according to their structural and genetic sequences. This phase is based on a language-independent approach which is decomposed into two steps:

1. A sequence alignment technique is applied between each pairs of structural sequences and each pairs of genetic sequences in order to achieve structural and semantic matching.
2. Two scores corresponding respectively to the structural and the semantic similarity between the compared codes are computed using the result of the structural and semantic matching.

A. *Sequence Alignment*

For the structural and the genetic alignments we employ the Dotplot technique [20] which was also used successfully in our audio retrieval system [21]. The similarity value between the sequences A and B is computed using the following equation:

$$Sim(A,B) = \max\left(\frac{\sum_i Seq_i^{A+}}{|A|}, \frac{\sum_j Seq_j^{B+}}{|B|} \right)$$

Where, SeqA+ and SeqB+ represent all the longest common sub-sequences without overlapping any inclusion between them.

B. *Fingerprint Matching*

The fingerprint matching is distilled into two levels. The first level called structural matching consists of producing a mapping between the similar salient components of the two compared codes according to the structural sequence similarities. Therefore the structural similarity between the codes *d* and *q* is computed as follow:

$$StructuralScore(d,q) = \frac{\sum_{i=1}^{N} Sim_i(S_{Ak}, S_{Aj})}{N}$$

Where:
- S_{Ak} and S_{Aj} represent the structural sequences of the k^{th} and j^{th} salient component within *d* and *q*.
- *N* represents the number of the best structural matching.

In the second level, the semantic matching consists of producing a mapping between each pair of similar salient elements within the compared codes according to the genetic sequence similarities. The semantic similarity between each pair of codes is computed as follow:

$$SemanticScore(d,q) = \frac{\sum_{i=1}^{M} Sim_i(S_{Bk}, S_{Bj})}{M}$$

Where:
- S_{Bk} and S_{Bj} represent the structural sequences of the k^{th} and j^{th} salient element within *d* and *q*.
- *M* represents the number of the best semantic matching.

V. Experiments and Evaluation

The system is implemented using the C++ language, and we used the Antlr tool [22] which is the appropriate solution to achieve the concept of Grammar-Actions.

In this section we describe our data sets and the evaluation of the robustness of the system against the different transformations that occur during the plagiarism and duplication of code. These transformations range from the very simple such as the simple Copy/Paste to the more complex such as alteration of the control flow as shown in Table II.

For our experiments, we constitute a collection of 533 source codes (200 Java sources, 143 C/C++ source and 190 CSharp sources). The dataset consists of 179701 lines (76183 for Java, 39167 for C/C++, and 64351 for CSharp). The dataset contain also the five source codes presented in Tables II to IV.

The code presented in Table III is a plagiarized version of the original code in Table II. The transformations that occur during the plagiarism operation are the modification of the variables names and the alteration of the control flow by grouping code portions in new functions (f1 and f2).

The original C++ code presented in Table VI is translated into CSharp and Java codes which are respectively presented in Tables V and IV.

In order to evaluate the relevance of the system, we query the five source codes against the dataset. The retrieval system returns the most similar codes matching the query code where the obtained results are illustrated in Table I.

TABLE I
RETRIEVAL RESULTS OF THE FIVE QUERY

TABLE II	TABLE II	TABLE III	
Structural score	100 %	86.95%	
Semantic score	100 %	100 %	
TABLE III	TABLE III	TABLE II	
Structural score	100 %	86.95%	
Semantic score	100 %	100 %	
TABLE VI	TABLE VI	TABLE V	TABLE IV
Structural score	100 %	95.83 %	70.83 %
Semantic score	100 %	100 %	81.48 %
TABLE V	TABLE V	TABLE IV	TABLE VI
Structural score	100 %	73.68 %	95.83 %
Semantic score	100 %	83.33 %	100 %
TABLE IV	TABLE IV	TABLE V	TABLE VI
Structural score	100 %	73.68 %	70.83 %
Semantic score	100 %	83.33 %	81.48 %

TABLE II
ORIGINAL CODE

```
1  int main(){
2  float x=-2.0,y=1.2,z;
3  z=fabs(x);
4  x = pow (x,2);
5  y++;
6  x+=y;
7  z=x+y;
8  printf("%f,%f,%f",x,y,z);
9  if (x>z)
10   y = y/z;
11 if ((y-1)>(y/z))
12   y--;
13 for ( ;x<200 ;x++)
14   z = z / y*x;
15 return 0;
16 }
```

TABLE III
PLAGIARIZED CODE

```
2 float l1=-2.0, O1=1.2, l11;
  int f1(){
3  l11=fabs(l1);
4  l1 = pow (l1,2);
5  O1++;
6  l1+= O1;
   return 0;
  }
  int f2(){
7  l11= l1 + O1;
8  printf("%f,%f,%f", l1 , O1 ,
l11);
9  if (l1> l11)
10  O1= O1/ l11;
11 if ((O1-1)>( O1 / l11))
12  O1--;
   while (l1<200){
14  l11= l11/ O1* l1;
    l1++ ;
    }
  }
1  int main(){
    f1();
    f2();
15  return 0;
16 }
```

TABLE V
C# CODE

```
1  #define DEBUG
   public class CSharpCode{
2  void f_1(ref float arg1){
3   arg1++;}
4  int f_2(ref double arg1){
5   arg1--;
6  return 0;}
7  public static void Main(){
8   float a=-2.0,b=1.2,c;
9   b++;
10  a+=b;
11 for (int j=0;j<12;j++)
12  c=a+b;
13 if (c>a+b)
14  c--;
15 #if DEBUG
16  Console.WriteLine("debug");
17 #endif
18  double argD = 0;
19  float  argF  = 0;
20  f_1(ref argF);
21 int res = f_2(ref argD);}
}
```

TABLE IV
JAVA CODE

```
class ArgDouble{
 public double val;
 public ArgDouble(double arg1){
 val = arg1;}
 }
class public JavaCode {
 static final boolean DEBUG = true;
 float f_1(float arg1){
  arg1++;
return arg1;}
 int f_2(ArgDouble arg1){
  arg1.val--;
  return 0;}
 public static void main(){
  float x=-2.0,y=1.2,z;
  y++;
  x+=y;
  for (int i=0;i<12;i++)
   z=x+y;
  if (z>x+y)
   z--;
  if (DEBUG)
   System.out.println("debug");
  ArgDouble argD = new ArgDouble(0);
  float  argF  = 0;
  argF = f_1(argF);
  int res = f_2(argD);}
}
```

TABLE VI
C++ CODE

```
#define DEBUG
void CppCode::f_1(float &arg1){
 arg1++;}
int CppCode::f_2(double &arg1){
 arg1--;
 return 0;}
void main(){
 float x=-2.0,y=1.2,z;
 y++;
 x+=y;
 for (int i=0;i<12;i++)
   z=x+y;
 if (z>x+y)
   z--;
 #ifdef DEBUG
  printf ("debug");
 #endif
 double argD = 0;
 float  argF  = 0;
 CppCode::f_1(argF);
 int res = CppCode::f_2(argD);}
```

VI. CONCLUSION

In this paper, we illustrated an efficient and robust source code retrieval system, which is based on structural-semantic fingerprint. The obtained results are very satisfying and particularly for the semantic level where the most queries have a similarity score more than 80%. Therefore, the structural sequences are less robust against the alteration of the control flow. The genetic and structural sequences are invariant to the translation of the programming language and especially to the translation from C++ to CSharp and from Java to CSharp.

REFERENCES

[1] M. Fowler and K. Beck, Improving the Design of Existing Code, Addison-Wesley Professional, 1999.

[2] J. Kerievsky, Refactoring to Patterns, Addison-Wesley Professional, 2004.

[3] J.-P. Retaillé, Refactoring des applications Java/J2EE, Eyrolles, 2005.

[4] E.L. Burd and M. Munro, "Investigating the Maintenance Implications of the Replication of Code", International Conference on Software Maintenance, IEEE Computer Society, Bari, Italy, 1-3 October1997.

[5] C. Kapser and M.W. Godfrey, "Toward a taxonomy of clones in source code: A case study", In Proceedings of the First International Workshop on Evolution of Large-scale Industrial Software Applications (ELISA), IEEE, September, 2003.

[6] S. Ducasse, M. Rieger, and S. Demeyer. "A language independent approach for detecting duplicated code", International Conference on Software Maintenance, IEEE Computer Society, Oxford, England, 1999, pages 109–118.

[7] B.S. Baker, "On finding duplication and near-duplication in large software system", Proceedings of Second Working Conference on Reverse Engineering, 1995.

[8] M. Halstead, "Elements of Software Science". Elsevier, New York, 1977.

[9] K. Ottenstein, "An algorithmic approach to the detection and prevention of plagiarism", ACM SIGCSE Bull, Vol 8, 1976, pages 30–41.

[10] J. Donaldson, A. Lancaster, and P. Sposato, "A plagiarism detection system", ACM SIGCSE Bull, vol 13, 1981, pages15–20.

[11] J.A. Faidhi and S.K. Robinson, "An empirical approach for detecting program similarity and plagiarism within a university programming environment", Computer Education, Vol. 11, 1987, pages 11-19.

[12] K. Verco and M. Wise, "Software for detecting suspected plagiarism: comparing structure and attribute counting systems", Proceedings of the First Australian Conference on Computer Science Education, In J. Rosenberg, editor, ACM Press, 1996.

[13] J.F. Sowa, "Conceptual structures: information processing in mind and machine", In Proceedings of the 1993 ACM/SIGAPP symposium on applied computing, ACM Press, 1993, pages 476–481.

[14] G. Mishne and M. Rijke, "Source Code Retrieval using Conceptual Similarity", Language & Inference Technology Group University of Amsterdam, 2004.

[15] M. Wise, "YAP3: improved detection of similarities in computer program and other text", In Proc. 27th SIGCSE Technical Symp. on Computer Science Education, Philadelphia USA, February 15–18, 1996, pages 130–134.

[16] L. Prechelt, G. Malpohl, and M. Philippsen, "Finding plagiarisms among a set of programs with Jplag", Technical Report No. 1/00, University of Karlsruhe, Department of Informatics, March 2000.

[17] A. Aiken, "MOSS: a system for detecting software plagiarism", University of Berkeley, CA, available http://www.cs.berkeley.edu/~aiken/moss.html,1998.

[18] C.A.R. Hoare, "Some Properties of Predicate Transformers", Journal of the ACM, 25(3), July, 1978, pages 461-480.

[19] M.A. Ouddan and H. Essafi, " Caractérisation de Documents Code Source Basée sur un Dictionnaire de Grammaire: Application à la Détection de Plagiats", International Conference on Sciences of Electronic, Technology of Information and Telecommunications, SETIT 2007, IEEE, Tunisia, 25-29 Mars, 2007.

[20] J. Helfman, "Dotplot Patterns: A Literal Look at Pattern Languages", TAPOS, 2(1), 1995, pages 31-41.

[21] M.A. Ouddan, S. Sayah, M. Taïleb and H.Essafi, "Audio Database Retrieval Based on Sequence Alignment", ICSES'06, International Conference on Signals and Electronic Systems, Poland 17-20 Septembre 2006.

[22] http://www.antlr.org/