Available online at www.sciencedirect.com

SCIENCE DIRECT®

Data & Knowledge Engineering 52 (2005) 249–271

DATA & KNOWLEDGE ENGINEERING

www.elsevier.com/locate/datak

Bio2X: a rule-based approach for semi-automatic transformation of semi-structured biological data to XML

Song Yang [a], Sourav S. Bhowmick [a,*], Sanjay Madria [b]

[a] *School of Computer Engineering, Nanyang Technological University, Singapore 639798, Singapore*
[b] *Department of Computer Science, University of Missouri-Rolla, Rolla 65409, USA*

Received 21 May 2004; received in revised form 21 May 2004; accepted 21 May 2004
Available online 1 July 2004

Abstract

Data integration of geographically dispersed, heterogeneous, complex biological databases is a key research area. One of the key features of a successful data integration system is to have a simple self-describing data exchange format. However, many of the biological databases provide data in flat files which are poor data exchange formats. Fortunately, XML can be viewed as a powerful data model and better data exchange format. In this paper, we present the *Bio2X* system that transforms flat file data into highly hierarchical XML data using rule-based machine learning technique. Bio2X has been fully implemented using Java. Our experiments to transform real world biological data demonstrate the effectiveness of the Bio2X approach.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Flat files; Rule base; Machine learning; XML; Transformer

1. Introduction

Technological advances in high throughput screening coupled with the genomic revolution resulted in a large amount of life science data. These data, which lie at the foundation of hypothesis development and experiment validation, are highly evolving, heterogeneous, semi-structured and not consistently represented. They are often stored in geographically distributed

* Corresponding author.
E-mail addresses: assourav@ntu.edu.sg (S.S. Bhowmick), madrias@umr.edu (S. Madria).

0169-023X/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2004.05.008

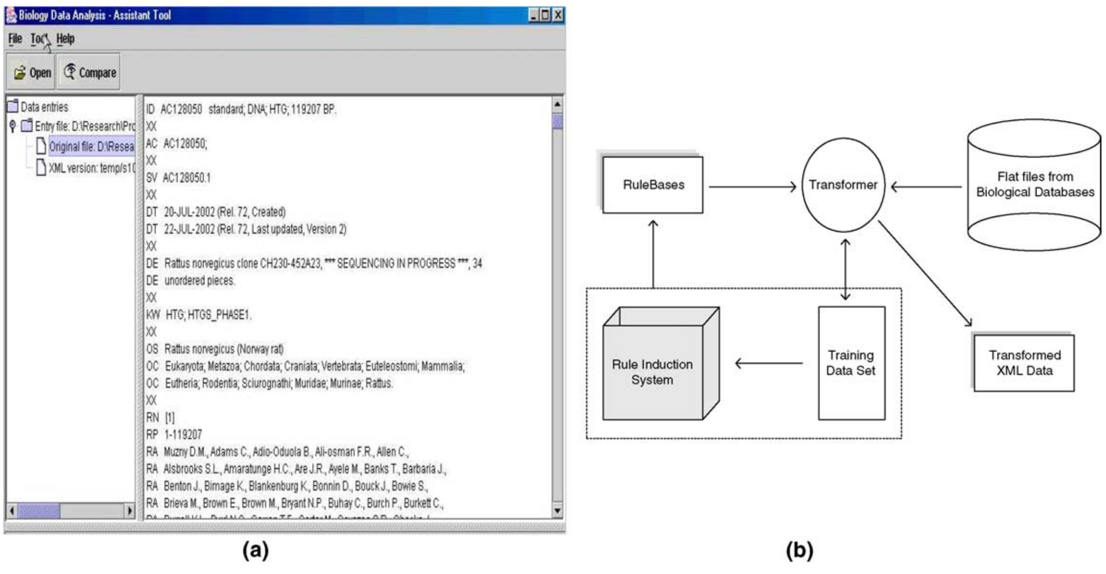
databases. These databases hold information that is critical to biomedical and life science researchers. In fact, it is realized that these data sources can be used in combination to answer queries that cannot be answered by any single data source. Thus a general data integration system that can handle heterogeneous, complex, and geographically dispersed biological data sources is a key area of research in bioinformatics.

As observed in [13], the main features of such data integration system are: data models that support nesting of structures, simple self-describing data exchange formats, thin wrappers, high-level query languages, good query optimizers, host language interfaces, and backend stores that support nesting of structures. Typically, the aspects of a self-describing data exchange format are: "lexical"—how to parse? "logical"—what is the structure? and "semantics"—what is the meaning? The lexical and logical layers are the important aspects of an exchange format. The better they are designed, the easier it is to exchange information. The semantics layer is not needed to accomplish the exchange of data between two systems [13]. In this paper, we focus our attention on the issue of self-describing data exchange formats.

1.1. Motivation

Most of the data provided by the online biological databases are semi-structured flat files by default. Moreover, each biological database has different format. Unfortunately, these flat files can be considered as poor data exchange format [13]. The EMBL data file in Fig. 1(a) is an example of such poor format. Some of the major problems of using such file format are as follows:

- SRS [8] is arguably the most widely used database query and navigation system for the Life Science community. In SRS, the data sources are generally required to be available as flat files and description of the source or schema must be available as *Icarus* script. Suppose a user wants to



```
ID  AC128050  standard; DNA; HTG; 119207 BP.
XX
AC  AC128050;
XX
SV  AC128050.1
XX
DT  20-JUL-2002 (Rel. 72, Created)
DT  22-JUL-2002 (Rel. 72, Last updated, Version 2)
XX
DE  Rattus norvegicus clone CH230-452A23, *** SEQUENCING IN PROGRESS ***, 34
DE  unordered pieces.
XX
KW  HTG; HTGS_PHASE1.
XX
OS  Rattus norvegicus (Norway rat)
OC  Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi; Mammalia;
OC  Eutheria; Rodentia; Sciurognathi; Muridae; Murinae; Rattus.
XX
RN  [1]
RP  1-119207
RA  Muzny D.M., Adams C., Adio-Oduola B., Ali-osman F.R., Allen C.,
RA  Alsbrooks S.L., Amaratunge H.C., Are J.R., Ayele M., Banks T., Barbaria J.,
RA  Benton J., Bimage K., Blankenburg K., Bonnin D., Bouck J., Bowie S.,
RA  Brieva M., Brown E., Brown M., Bryant N.P., Buhay C., Burch P., Burkett C.,
```

Fig. 1. EMBL data and Bio2X. (a) EMBL screenshot and (b) architecture of Bio2X.

specify a query that finds all the EMBL entries from the division *invertebrates* having sequences with *length* greater than 300 and that have a direct link to enzymes characterized in the ENZYME database. It is difficult to execute such queries as SRS has limited data joining capabilities over flat files. In fact, many complex queries is almost impossible to formulate in SRS efficiently [17].

- Consider the above query again. Suppose the query results must return the EMBL accession numbers, descriptions and the nucleotide sequences. However, due to very different data format of each biological source, it is not easy to collect such relevant information from different online biological databases (EMBL and Enzyme) and restructure them into a single structured file.
- It is also important to be able to conveniently warehouse the data locally [17]. A commercial relational database system is an attractive option for warehousing very large amount of data due its simplicity, scalability, stability and expressiveness. However, transforming flat files directly to relational database system forces us to unnaturally and unnecessarily fragment our data in order to fit our data into the third normal form [17], which increases the cost of certain queries significantly due to the large number of joins.
- It is of immense importance in bioinformatics to have the ability to use the results of some analysis as the basis for conducting further downstream analysis (e.g. identifying new genes) in a manageable, efficient way. However, with the diverse range of data formats and different computer platforms, it has become an increasingly daunting task to work with different analysis tools. Researchers wishing to perform multiple-pass analysis of data, by feeding results of one program to another, encounter the problem of changing data from one format to another.

1.2. XML as data exchange format

XML, on the other hand, allows for a hierarchical nesting of tags and the set of tags can be defined flexibly. XML is self-describing (describes the data itself) and the logical and lexical layers are separated so that a generic parser can be used for any data laid out in this exchange format. The logical layer conforms to the nested structure of biological data also. Thus, XML can be viewed as a powerful data model and better exchange format compared to flat files, providing directly for two of the important ingredients of a general data integration solution in bioinformatics [17]. Consequently, the limitations discussed in Section 1.1 can be addressed due to the following reasons.

- First, a set of XML documents can be regarded as a database and can be directly processed by a database application or queried via powerful XML query languages. This provides the means for querying across multiple data sources, filtering the documents based upon the contents, and restructuring the results into a more suitable form for subsequent analysis steps.
- Second, recent research [16] demonstrates that it is indeed possible to use standard commercial relational database systems to store, index and evaluate powerful queries over XML documents. This means that all of the power of relational database systems can be brought to bear upon to process queries in the local warehouse efficiently.
- Third, XML may be used to resolve the problem of multipass analysis by providing one consistent format for wide variety of tools and databases.
- Finally, genomic data are increasingly being provided in XML format. New bioinformatics applications also increasingly take XML as input, making it essential that existing non-XML

data be easily converted to XML. For instance, PIR, Entrez are already becoming XML compatible [1]. Several public domain and proprietary XML databanks such as the INTERPRO databank are already in existence.

Hence, it is crucial to use *wrapper* technology to extract the relevant information from flat files and translate it into XML which can be easily queried or further processed. In this paper, we present the design and implementation of Bio2X that converts flat files into highly hierarchical XML form based on *extraction rules*.

1.3. Overview of Bio2X

Fig. 1(b) depicts the architecture of Bio2X. Bio2X currently converts flat file data from GenBank, EMBL, Swiss-Prot, and PDB into XML data. Because of the disparities of the file structures of different biological databases, we define a different set of rules for each database. However, the rule bases are designed in a consistent manner so that a single transformer is sufficient to parse any data file from any database. The transformer will choose a suitable rule base for parsing the input flat file based on its origin database and generate the XML data file. The rule base exploits the *hierarchical* structure of the source to constrain the data extraction problem. It allows for extraction of target patterns based on surrounding landmarks, *line types* and other lexical patterns in flat files. It also allows for more advanced features such as disjunctive pattern definitions. Finally, it involves machine-learning techniques to refine the rules in order to improve the accuracy of the transformation. A working prototype of Bio2X has been implemented using Java. Preliminary results on representative flat files using the current Bio2X prototype show a good performance (see Section 5).

The rest of the paper is organized as follows: Section 2 describes the structure of biological data. Section 3 discusses the design of extraction rules. In Section 4, we illustrates the rule induction system. Section 5 presents some experimental results of Bio2X. We discuss related approaches in Section 6. Finally, the last section highlight future research directions.

2. Structure of biological data

Most of the data provided by the online biological databases are flat files by default. As each database has different format, we use EMBL nucleotide sequence database as our running example to highlight the semi-structured nature of biological data. The EMBL Nucleotide Sequence Database constitutes Europe's primary nucleotide sequence resource. Main sources for DNA and RNA sequences are direct submissions from individual researchers, genome sequencing projects and patent applications. Fig. 1 shows a partial view of a sample entry from the EMBL database (due to space constraints we only present a partial view of the data).

Observe that each entry in the data file is composed of lines. The general structure of a line in EMBL is given in Fig. 2. Each line in the data file is defined by a *line code* and a *value*. A *line code* represents the type of information (*line type*) specified in the line. The line types, along with their respective line codes, are listed in Fig. 3. Note that some entries do not contain all of the line types, and some line types occur many times in a single entry.

Characters	Content
1-2	Two character line code.Indicates the type of information contained in the line
3-5	Blank
6-78	Data

Fig. 2. Structure of a line.

Code	Type	Description
ID	Identification	Begins each entry, 1 per entry
AC	accession number	>=1 per entry
SV	new sequence identifier	>=1 per entry
DT	date	2 per entry
DE	description	>=1 per entry
KW	keyword	>=1 per entry
OS	organism species classification	>=1 per entry
OG	organelle	>=0 or 1 per entry
RN	reference number	>=1 per entry
RC	reference comment	>=0 per entry
RP	reference positions	>=1 per entry
RX	reference cross-reference	>=0 per entry
RA	reference author(s)	>=1 per entry
RT	reference title	>=1 per entry
RL	reference location	>=1 per entry
DR	database cross-reference	>=0 per entry
FH	feature table header	>=0 or 2 per entry
FT	feature table data	>=0 per entry
CC	comments or notes	>=0 per entry
XX	spacer line	many per entry
SQ	sequence header	1 per entry
BB	(blanks) sequence data	>=1 per entry
//	Termination line	Ends each entry

Fig. 3. Line types and their codes.

Let us now define a few terms for our exposition. Each line type in an entry is called a *bio-attribute*. Each bio-attribute may have a string *value* which may be empty. For example in the third line of Fig. 1, *AC* is a bio-attribute and *AC128050* is the corresponding value. Hence, an entry can be considered as a list of bio-attribute–value pairs. Each such pair is collectively called as *bio-element*.

The bio-elements in the flat file can also be classified into the following three types based on the different logical structure of the values of bio-attributes:

- *Fixed format bio-elements*: Sequence data (SQ) and feature table (FT) lines have fixed format. The sequence data is written 60 bases per line, in groups of 10 bases separated by a blank character, beginning in position 6 of the line. The feature table contains information about genes

```
CC   Contact: Sanchez DO
CC   Genomics and Bioinformatics
CC   Instituto de Investigaciones Biotecnologicas
CC   Parque Tecnologico Miguelete,INTI, Edificio 24, 1650, San Martin,
CC   Buenos Aires, Argentina
CC   Tel: (54-11) 4580-7255
CC   Fax: (54-11) 4752-9639
CC   Email: dsanchez@iib.unsam.edu.ar
CC   Sequences were base called with phred and vector was masked with
CC   crossmatch (see http://www.phrap.org). Sequences were then trimmed
CC   from both ends to remove low quality bases and masked vector.
CC   Plate: 01 row: a column: 3
CC   Seq primer: T7.
```

Fig. 4. A sample comment section.

and gene products, as well as regions of biological significance reported in the sequence. Each feature consists of a descriptor line containing a *feature key* and a *location*. One or more lines containing feature qualifiers may follow the descriptor line. The *feature qualifier* always starts with a '/' mark, and its value is stated after an equal sign.

- *Standard pattern bio-elements*: Elements such as *keywords* (*KW*), *reference authors* (*RA*) do not have a fixed format. However, they may contain some standard lexical patterns in terms of position of items on a line, and the delimiters of a value list, etc. For example, the *keywords* (KW) of an entry may span multiple lines. More than one keyword may be listed on each KW line; the keywords are separated by semicolons, and the last keyword is followed by a period. For example, in Fig. 1, `KW HTG; HTGS _PHASE1`. Note that such regulated format makes it easy to extract each keyword out of the keywords collection based on the separator ';'. (Discussed in Section 3.)
- *Free-text bio-elements*: Values of some bio-attributes are highly unstructured. For example, consider the *comment* (CC) bio-element in Fig. 4. It can be observed that this comment segment is actually composed of several types of information such as contact, telephone number, email, fax number, plate etc. There is also a segment of miscellaneous information highlighted in bold in Fig. 4. It is desired to extract those information from the *comment* bio-element to form separate hierarchical XML elements in the transformed XML. Observe that several parts in Fig. 4 consist of a *title* and a value, which are separated by a colon. For example, *Email* is the title having value equal to dsanchez@iib.unsam.edu.ar. Such observations can be used as a rule to isolate and extract information. However, note that there is no general rule for extraction from all such data entries. This is because: (1) The title does not always start from the beginning. (2) The title and its value are not always separated by a colon. (3) Some colon may be part of a string instead of acting as a separator, such as the colon in the URL (http://www.phrap.org) in Fig. 4. (4) It is difficult to determine terminating point of value data corresponding to a title. (5) Some information may not have a title as shown in bold in Fig. 4. It is interesting to note that all the potential problems associate with free-text elements and solutions cannot be fully foreseen. So, it is desired to have an intelligent rule system that can be dynamic and refined

automatically to improve the accuracy of conversion of flat files to XML documents. We discuss how we achieve this in detail in the subsequent sections.

3. Design of extraction rules

Recall that a flat file can be considered as a list of bio-attribute–value pairs, and values of many bio-attributes in most of the biological databases are not atomic and can be broken down further into subattribute–value pairs. Consequently, it would be desirable to represent such subattributes as hierarchical structures in the XML, rather than putting an entire bio-element as a child element of the root. For example, the *dates* bio-attribute in Fig. 1 has two types of dates (creation date and last updated date) stored in its value. Furthermore, the release and version numbers of the entry are also encoded in the value of the date. Consequently, Bio2X should be able to extract various information and represent them appropriately in the transformed XML document.

This is a challenging task as the values in the flat files can be highly semi-structured or unstructured as discussed in Section 2. Consequently, automatic extraction of subattribute–value pairs from these unstructured components of the flat files is a non-trivial problem. In this section, we address this issue in detail.

3.1. Overview

Our approach exploits the *hierarchical* structure of the source to constrain the data extraction problem. More precisely, based on the structure of biological data in the flat files, we *automatically* decompose one difficult problem (extract all bio-attributes of interest) into a series of simpler ones. For instance, instead of using one complex rule that extracts all *reference titles* from an entry, we take a hierarchical approach. First, we apply a rule that creates necessary tags for representing multiple *reference* elements in the transformed XML. Second, we use a rule that extracts the list of *references*; then we use another rule to break the list into items that correspond to individual references; finally, from each such item we extract the *reference title* of the corresponding reference information and create necessary elements/attributes in the XML document to store the information. We now discuss these steps in detail. First, we discuss how to create the children of the root element of XML tree (first step) by examining the *line types*. Then, we present how to further extract hierarchical structure from the values of the bio-elements in the flat file (second and third steps).

3.2. Extracting children of the root

For simplicity of illustration, assume that the EMBL data file is composed of three line types: *dates* (DT), *references* (RX), and *comments* (CC). We first define a set of rules (Fig. 5) to isolate each of the three bio-elements and form the children of the root node in the XML tree. Further extraction of data information is localized within each segment in order to reduce the complexity of locating the data. The rules for further extraction are incorporated into the basic rule set, which will be discussed in the following section.

The extraction rules are formulated in XML format. The justification for using XML format is two folded: first, it exhibits a consistent tree structure with the XML data file to be generated. The

```
<?xml version="1.0" encoding="UTF-8"?>
<entry>
   <attr>
      <name>db</name>
      <value>embl</value>
   </attr>
   <element multiple= "F">
      <tag>DT</tag>
      <name>dates</name>
   </element>
   <element multiple= "F">
      <name>references</name>
      <element multiple= "T">
         <tag>RN</tag>
         <name>reference</name>
      </element>
   </element>
   <element multiple= "F">
      <tag>CC</tag>
      <name>comment</name>
   </element>
<entry>
```

Fig. 5. Rules for formulating the first level elements.

parent–child relationship is represented by the markup rule elements. Second, the rule can be easily loaded into a tree structure by a DOM parser. So, it is easier for a transformer to get the rule information rather than reading from a flat file. Let us now elaborate on the key markups in Fig. 5.

The root element *entry* has four child nodes: one *attr* and three *element* nodes. The XML element *attr* is used to describe the attribute of the root node. The *element* node represents the child of the root entry in the transformed XML. They may also contain their child *element* markups recursively to form a multi-level tree structure. The markup *tag* indicates the line code to be detected in the flat file and the markup *name* specifies the name to be used for the XML element in the transformed XML document for the corresponding bio-element. The attribute *multiple* has a boolean value and is used to specify whether the element can occur multiple times in the XML document. For example, consider the *reference data* (RN) in Fig. 1. Biological data may have several references, and each reference data is identified by a new RN line. So, the rules for extracting reference data is formed by: first, specify an element called *references* without any tag information in order to encapsulate all *reference* elements. Second, locate an element *reference* by detecting the line code RN. The attribute *multiple* is set to 'T' which means that multiple occurrences of RN are possible. The same set of rules will be applied to all occurrences of *reference* elements.

After wrapping with the basic rules listed in Fig. 5, the flat file is transformed into an hierarchical XML structure shown in Fig. 6. Next, we restrict the scope of data extraction within each element in the XML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<entry db="embl">
   <dates/>
   <references>
      <reference/>
   </references>
   <comment/>
<entry>
```

Fig. 6. The XML structure after segmentation.

3.3. Extracting hierarchical structure from values

Consider the simplest example of extracting data from *dates* (DT) in Fig. 1. The XML version is shown in Fig. 7. The rules for performing this transformation are listed in Fig. 8, in which the rules for extracting the *last-updated* information are omitted to simplify visualization. Note that the *dates* element in Fig. 5 has been expanded to incorporate these rules. As the values of *dates* bio-element contain two types of dates: *creation* and *last_updated*, two corresponding XML elements exist as well in the transformed XML, which are composed of attribute values. The name of an attribute is specified by the *name* markup. Its value is located with the rules specified under *loc* markup (Fig. 8).

The *loc* element precisely defines how to locate the information within the bio-element. For any given item to be extracted, one needs an extraction rule to locate both the beginning and end of the item. Since, in our framework, each value in a bio-element consists of a sequence of tokens (words, punctuation, etc.), this is equivalent to finding the token appearing just before the item and the token appearing immediately after the item (these tokens are called *landmarks*). In this example, to extract the creation date, the starting landmark is empty (⟨`start/`⟩), meaning that the value starts from the beginning of *dates* element. The end landmark is '(', meaning that the date item should extend until a token '(' is met. Similar technique is used in other *loc* elements.

Note that it is not always judicious to use such landmarks for any biological data extraction. For example, consider the extraction of keywords from a keyword list: KW 28S ribosomal RNA; 28S rRNA gene. Since the number of keywords appearing in a data file varies, it is not possible to specify the starting and ending landmarks for each keyword to be extracted. The simplest way of

```
<?xml version="1.0" encoding="UTF-8"?>
<entry db="embl">
   <dates>
      <created date="20-JUL-2002" release_num="72"/>
      <last_updated date="22-JUL-2002" release_num="72" version_num="2"/>
   </dates>
<entry>
```

Fig. 7. Transformed XML fragment.

```
<?xml version="1.0" encoding="UTF-8"?>
<entry>
   <element multiple="F">
      <tag>DT</tag>
      <name>dates</name>
      <element>
         <name>created</name>
         <attr>
            <name>date</name>
            <loc type="string">
               <start/>
               <end>(</end>
            </loc>
         </attr>
         <attr>
            <name>release_num</name>
            <loc type="string">
               <start>Rel.</start>
               <end>, Created)</end>
            </loc>
         </attr>
      </element>
      .......
   </element>
<entry>
```

Fig. 8. Rules for extracting data from dates.

overcoming this is to use the special token ';' as a separator for tokenizing the keyword list. This rule is specified in Fig. 9.

Hence, it is necessary to distinguish between the different types of data extraction. In our approach we use an attribute field called *type* in *loc* element to distinguish the different ways of extracting data. This is very flexible as any number of new types can be added in order to extract any type of data. A summary of some major *types* in Bio2X is shown in Fig. 10.

Besides the simple utilization of landmarks with different types involved, some more complex constructs are also needed for data extraction. These extensions are introduced in order to use

```
<?xml version="1.0" encoding="UTF-8"?>
<entry>
   <element multiple="F">
      <tag>KW</tag>
      <name>keywords</name>
      <loc type=";_list_.">
         <name>keyword</name>
      </loc>
   </element>
<entry>
```

Fig. 9. Rules for extracting keywords.

Type of Loc	Description
string	Use starting and end landmarks to extract string.
integer	Use starting and end landmarks to extract integer.
pos	The start and end markups specifies the starting and ending position of an item on a line.
,_list	Indicate that the element is a list of items separated with comma. Only name name markup is specified under loc to specify the name of each item extracted from the list.
match	Contains a set of value elements. The result of extraction will match with one of the values.

Fig. 10. Summary of major types of *loc* element.

```
RX MEDLINE; 22200211.
RX PUBMED; 12208047.


<reference_info db="MEDLINE" id="22200211">
<reference_info db="PUBMED" id="12208047">
```

Fig. 11. RX data and transformed XML fragment.

minimal number of rules to express all the necessary lexical patterns appearing in biological data. For example, consider the RX data and its transformed XML version in Fig. 11. The key rule here is that every line of RX data is to be segmented into an individual item. Then, further extraction of *db name* and *id number* is restricted within the local scope of each line. This is accomplished by rule markups *list* and *separator*. Different separator values can be used to segment a whole element into numerous parts, and each part will be treated individually to extract data from within. This provides a hierarchical approach for extracting data from the current scope only. Fig. 12 depicts an example of such rules.

3.4. Disjunctive rules

Another key feature of our extraction rules is that it allows the use of disjunctions by using the markup *choice* in the rule definition. It enables multiple rules to be specified for extracting the same data, and enforces an either–or relationship between the rules. This is useful when a single rule is not capable of extracting relevant data from all the possible cases. For example, in the case of extracting data from *organism_species* (OS), the OS line may appear in two formats: `OS scientific name (name)` or `OS scientific name`. In Fig. 1 OS follows the former format. In order to extract the scientific name (`Raitus norvegicus` in Fig. 1) out of OS line, a rule needs to state the start and end tokens for it. The start token is obviously empty since the name follows OS tag immediately. However, the end token may be a special character '(' or end of line.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<entry>
   <element multiple="F">
      <name>references</name>
      <element multiple="T">
         <tag>RN</tag>
         <name>reference</name>
         <element>
            <tag>RX</tag>
            <list>
               <separator>**</separator>
               <name>reference_info</name>
               <attr>
                  <name>db</name>
                  <loc type="string">
                     <start/>
                     <end>;</end>
                  </loc>
               </attr>
               <attr>
                  <name>id</name>
                  <loc type="string">
                     <start/>
                     <end>.</end>
                  </loc>
               </attr>
            </list>
         </element>
      <element>
   </element>
<entry>
```

Fig. 12. Rules for extracting data from RX.

Applying a disjunctive rule set is a straightforward process: the transformer successively applies each disjunctive rule in the list until it finds the first one that extracts some data. The transformer does not measure the correctness of the data extracted, so the order of the two rules is significant. In this case, the first rule with end token '(' has to be in front, because if the second rule is put in front, the transformer will always terminate after applying it without considering the other rules. The rule set for the above example is shown in Fig. 13. Note that the rules are not ordered arbitrarily. The most restricted rule should appear first, which means that the rule with fewer chances of extracting data should be parsed first in order to extract data from the relatively special cases. This also promotes the consideration for *perfect disjunction*, which means that the rule in the disjunctive list should either produce no data or the data extracted should be correct. These issues will be discussed further in the next section.

Our rule grammar is universal to all four biological databases (EMBL, Genbank, Swiss-Prot and PDB) we have explored with. With different combination and values of the rule elements,

```
<?xml version="1.0" encoding="UTF-8"?>
<entry>
   <element multiple="F">
      <tag>OS</tag>
      <name>organism_species</name>
      <element>
         <choice>
            <name>organism</name>
            <loc type="string">
               <start/>
               <end>(</end>
            </loc>
         </choice>
         <choice>
            <name>organism</name>
            <loc type="string">
               <start/>
               <end/>
            </loc>
         </choice>
      </element>
   </element>
<entry>
```

Fig. 13. Rules for extracting data from OS.

precise rule bases can be developed for each database. Next, a discussion of how to generate the rules is presented.

4. Rule induction system

Recall that the bio-attributes can be classified into three types based on the logical structure: fixed format, standard pattern, and free-text bio-elements. The rules of extracting data from fixed format or standard pattern bio-elements can be predefined based on the lexical patterns of the biological data files, such as the segmentation based on the line code, and extraction of keywords from keyword collections, etc. These rules guarantee the correctness of extraction as long as the formats of the biological databases do not change. However, the free-text bio-elements have very loose structure so that it is impossible to predefine a set of rules that works for all data files, such as the *comment* data.

4.1. Algorithm learnrule

A rule induction system that learns the extraction rules based on training data is designed to extract data from free-text element. A rule set is first defined by the user for data extraction. It is capable of extracting all the regular data based on the known lexical patterns. Then it is extended with a greedy-covering procedure [4]:

- The transformer takes the input data and converts it into XML based on the existing rules.
- The system verifies the XML file generated by comparing it with a correct XML file in the training set.
- If they are identical, then the training program continues for next data. Otherwise, the rules are refined to cover the training data.
- The above steps are repeated until all training data has correctly been transformed.

The rule set is refined by adding a new rule with a set of new landmarks whenever necessary. The new tokens are selected based on the following heuristics:

- The tokens nearest to the desired data are attempted as landmarks first.
- The punctuation symbols and special characters are chosen as the landmarks first.
- Spaces are treated as delimiters when trying to use words as landmarks.
- Combinations of punctuation and words are also tried. But the number of words and distance of punctuation are limited because of the semantic feature of English text and performance issue.

A rule needs to have the following two properties: first, it produces the correct extraction result if some data is extracted based on it. Second, it fails to extract any data on all other data files. However, this is difficult to achieve. A rule may produce correct results on some data files while producing wrong results on other data files. There may also be no ways of refining the rule any further. Hence, the solution is to order the rules to ensure the correctness of wrapping when applying the rules sequentially. The heuristics for ordering are listed with highest priority first [15]: (1) The rule with fewer early and late matches (i.e., mismatches) should appear earlier. (2) The rule with more correct matches is preferred to the other ones. (3) The rule with fewer tokens should appear earlier. The outline of the rule induction algorithm is shown in Figs. 14 and 15. However, the ordered rule set still does not guarantee the correctness of data extraction for the entire database. Although the newly generated rule will cover at least one example, yet it may produce erroneous result on some other data files. The performance will mainly depend on the selection of the training data.

Because of the advantages and limitations of rule-induction, our system employs rule-induction for data extraction from *comment* data mainly, which we believe is the major free-style text segment in biological databases. However, such learning technique can be easily extended to other bio-elements if there is any change in format in the future.

4.2. A case study

Let us now consider an example of rule induction for extracting relevant pieces of data from the *comment* fragment. Reconsider the *comment* fragment in Fig. 4. One can observe that the *comment* data is composed of a list of *title-value* pairs, in which colons separate the title and value, such as the *contact information*, *telephone number*, etc. There is also a segment of data (highlighted in bold

```
Input: Example $S$
Output: Ordered Rule Set $R$

predefine a set of rules $R$;
while ($S$ is not empty) {
    get the next example $E$ in $S$;
    wrap $E$;
    if (inconsistency occurs) {
        $rule = Refine(S)$; /* Figure 15 */
        add $rule$ to $R$;
    }
}
sort $R$ according to number of mismatches;
sort $R$ according to number of correct
matches;
sort $R$ according to number of tokens;
```

Fig. 14. Algorithm *LearnRule*.

```
Input: Example $S$
Output: New rule $R_n$

let $C$ be the tokens in $S$;
let $R_n$ be the new rule;
let $R_s$ be the trial rule set;
for $i = n\ to\ 1$ do {
    for each $R_n \in R_s$ {
        push $C[i]$ to $R_n$;
        if $C[i]$ is a punctuation
        or special character or space {
            test $R_n$ on $S$;
            if success return;
        }
        else {
            drop the last token in $R_n$;
            add $R_n$ to $R_s$;
        }
    }
}
get the last space in $C$ at position $x$;
set $R = C[x]C[n]$;
```

Fig. 15. Algorithm *Refine*.

in Fig. 4) with only value but no title. Hence, the major task in data extraction is to partition the data into *title–value* pairs (title may be empty).

First, consider the simplest case of extracting data related to *contact information*, *telephone number* and *fax number*. Because of the common pattern of "*title:value*", a colon is used as the

```
<?xml version="1.0" encoding="UTF-8"?>
........
        <information title = "Email">
            dsanchez@iib.unsam.edu.ar Sequences were base called with phred
            and vector was masked with
        </information >
        <information title = "crossmatch
        (see http">
            //www.phrap.org). Sequences were then trimmed from both ends
             to remove low quality bases and masked vector.
        </information >
```

Fig. 16. Erroneous comment transformation.

separator to partition the comment data. Once a colon is detected, the basic rule is to *extract the segment from the start of the line containing the colon until the start position of the next segment*. The tokens before the colon in the current segment are the *titles*, and the tokens after the colon sign are the values of the titles.

However, when the above rule is applied to extract the email information, an error will occur as shown in Fig. 16. The information in bold is erroneously divided into two parts. This occurs because the colon appearing in the web address (http://www.phrap.org) is mistakenly treated as a separator. If the colon in the web address is simply ignored, then the XML fragment generated will be as depicted in Fig. 17. This is due to the fact that the item next to email address does not contain a colon separator to indicate the existence of a new segment. As a result, the email segment will continue to extend until the starting of next recognizable segment (*Plate*). The rule induction system will detect these errors by comparing the result with a correct XML file. It refines the rule set by creating some new rules as shown in Fig. 18. The tokens are selected based on the heuristic that uses words as landmarks as there is no punctuation or special character before the colon. Consequently, the following new grammars are added into the rule base to facilitate the data extraction:

- `comment_pattern`: Records the data title and the end token for this segment. In this case, the title is *Email*, and the end token is system line separator. It also indicates that the next information following the current one has no information title. So, the next element will be extracted by getting the list of tokens following the current segment but before the next-next segment.

```
<?xml version="1.0" encoding="UTF-8"?>
........
        <information title = "Email">
            dsanchez@iib.unsam.edu.ar Sequences were base called with phred
            and vector was masked with crossmatch (see http://www.phrap.org).
            Sequences were then trimmed from both ends to remove low
            quality bases and masked vector.
        </information >
```

Fig. 17. Erroneous comment transformation.

```
<?xml version="1.0" encoding="UTF-8"?>
........
<comment_pattern>
   Email__**
</comment_pattern>
<ignore>
   http
</ignore >
<comment_tag>
   row
   column
</comment_tag>
```

Fig. 18. Rules for comment extraction.

- `ignore`: Indicates that any colon following the element will not be treated as a segment separator. In this case, a colon following *http* is part of web address, so it is ignored.

Finally, consider the transformation of the last line in Fig. 4. The XML generated should have the following three elements: *plate*, *row* and *column*. However, using the existing rules, the XML data will be as follows: `<information title = “Plate”> 01 row: a column: 3 </information>`. This is because there is no match with any element under `<comment_pattern>` and `<ignore>`. So, the transformer will not apply special techniques required by these two patterns. As a result, the transformer will apply the basic rules for extraction.

The rule induction system is responsible to handle the above problem. It discovers that two more elements are actually present (*row* and *column*). Hence, it adds the rules for specifying the titles as depicted in Fig. 18 using `comment_tag`. The new rule markup `comment_tag` is used to record the special comment titles that have to be located by string matching. In this case, after inserting the above rules, the transformer will generate correct results. In other situations, the whole induction process will be repeated again until the correct result is obtained. Fig. 19 shows the final transformed XML fragment of the comment segment in Fig. 4.

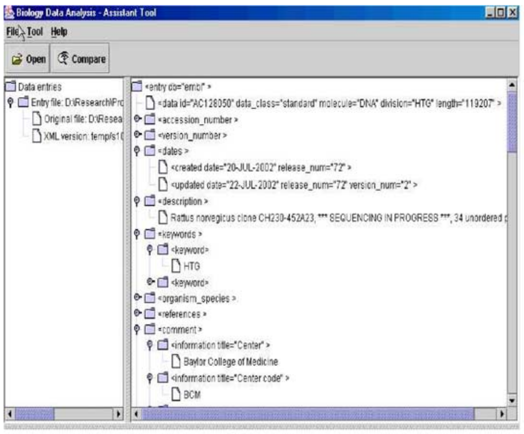
From our experience with various test data in the four databases, it is found out that these extra three markups for rules (`<comment_pattern>`, `<comment_tag>`, and `<ignore>`) are sufficient to segment the *comment* data. They have higher priority than the basic separator colon, so the special cases are handled first to enhance the accuracy of transformation.

5. Experimental results

We have implemented Bio2X in Java using J2SDK 1.4.0. Fig. 20 shows the screenshot of the transformed XML of the data in Fig. 1. Our transformer chooses a suitable rule base for the input flat file based on its origin database. Next, we parse the XML rule bases into a DOM tree. The tree nodes are then traversed in a top-down approach and the rule elements are parsed to derive the target and extraction method. The flat file is read sequentially at the same time for extracting data based on the information derived from the corresponding rule elements. The DOM tree and flat file are traversed only once without duplicate processing so the time complexity of translation

```
<?xml version="1.0" encoding="UTF-8"?>
........
   <comment>
      <information title= "contact" >
         Sanchez DO Genomics and Bioinformatics
         Instituto de Investigaciones
         Biotecnologicas Parque
         Tecnologico Miguelete, INTI,
         Edificio 24, 1650, San Martin,
         Buenos Aires, Argentina
      </information >
      <information title= "Tel" >(54-11) 4580-7255 </information >
      <information title= "Fax" >(54-11) 4752-9639 </information >
      <information title= "Email" > dsanchez@iib.unsam.edu.ar </information >
      <information>
         Sequences were base called with phred and vector was masked with
         crossmatch (see http://www.phrap.org). Sequences were then trimmed
         from both ends to remove low quality bases and masked vector.
      </information >
      <information title= "Plate" >01 </information >
      <information title= "row" >a </information >
      <information title= "column" >3 </information >
      <information title= "Seq primer" >T7 </information >
   </comment>
```

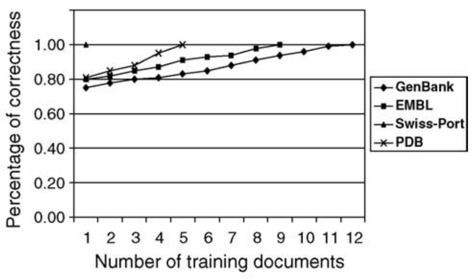
Fig. 19. Comment transformation.



```
Data entries
  Entry file: D:\Research\Pr
    Original file: D:\Resea
    XML version: temp/s1(

<entry db="embl" >
  <data id="AC128050" data_class="standard" molecule="DNA" division="HTG" length="119207" >
  <accession_number >
  <version_number >
  <dates >
    <created date="20-JUL-2002" release_num="72" >
    <updated date="22-JUL-2002" release_num="72" version_num="2" >
  <description >
    Rattus norvegicus clone CH230-452A23, *** SEQUENCING IN PROGRESS ***, 34 unordered
  <keywords >
    <keyword>
      HTG
    <keyword>
  <organism_species >
  <references >
  <comment >
    <information title="Center" >
      Baylor College of Medicine
    <information title="Center code" >
      BCM
```

(a)



(b)

Fig. 20. Experimental results. (a) A partial snapshot of transformed XML. (b) Correctness vs. size of training set.

is proportional to the file size only. To evaluate the performance of Bio2X, we collected 100 flat files from each of the four databases (EMBL, GenBank, Swiss-Prot, PDB). We ran the experiments on a Pentium III 900 MHz PC with 256 MB RAM under MS Windows 2000 Professional. We carried out three sets of experiments.

The first set of experiments demonstrates the effectiveness of Bio2X in the transformation process. We measure the accuracy rate of transformation. A pool of examples of each biological database is provided to the system for testing. The transformer translates each flat file into XML format and examines the correctness of data extraction. If any inconsistency occurs then both the generated XML file and the desired XML information are passed as training data to the rule induction system to refine the rule base. The criticality of the examples hence determines the effectiveness of transformation. Therefore, instead of random examples, the system is provided with carefully selected examples, which are highly informative and illustrate exceptional cases. To select the informative examples, we first download tens of thousands flat files from a biological database's FTP server. It is observed that the biological entries are submitted by several institutes only, and each institute normally arranges the information in a constant style even for the free-text bio-elements. Hence, we choose several flat files from each institute as representatives to preserve the variety of file formats and reduce the pool size of examples as well. In our experiment, 100 testing cases are selected for each biological database, and each testing case exhibits some unique feature from others. The accuracy rate over the testing cases is gradually improved along with rule induction from the training data. As the size of training data set increases, the correctness of transformation also increase until it reaches 100% accuracy, as shown in Fig. 20(b). Observe that Swiss-Prot achieves 100% accuracy by just using one file as training data. This is because the structure of Swiss-Prot is very regular. Consequently, its rule base can be completely predefined without any needs for rule refinement. The remaining three databases has varying degree of initial accuracy (75–81%) based on the predefined rule base. Observe that GenBank, EMBL and PDB takes 12, 9 and 5 data files respectively to achieve 100% accuracy by refining the rule set using rule induction process.

Note that the accuracy rate may drop when a larger set of testing cases are executed, since there may be some special cases that have not been covered by an extraction rule. When the special case is encountered, the rule induction system can learn a new rule from it to restore the accuracy rate again. As a result, the number of training documents will be incremented to achieve the full correctness, and the time taken for rule induction will be longer.

The second set of experiments investigates the time spent on rule induction process as shown in Fig. 21(a). Again, due to the regularity in structure of Swiss-Prot data, time taken for training is zero. The training for PDB takes longer time since its file size is much larger than that of GenBank and EMBL.

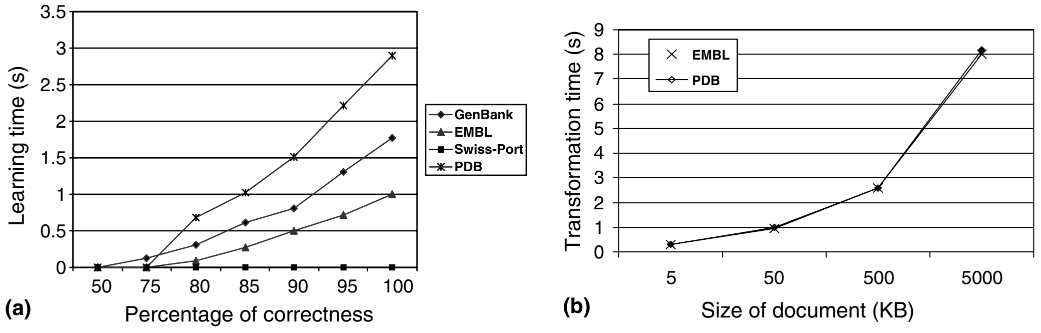


Fig. 21. Experimental results. (a) Learning time vs. correctness. (b) Time for XML transformation.

The third experiment illustrates the time taken to transform to XML by as the size of the flat files in different databases increases (Fig. 21(b)). We only illustrate EMBL and PDB as the time taken by all the four databases are almost identical when the file size is equal. It is evident from Fig. 21(b) that the time required for XML transformation is proportional to the size of the document.

6. Related work

Our proposed biological data transformation system is largely influence by several recent technologies by three research communities. The bioinformatics community has looked at developing good data exchange formats for representing biological data in flat files that would facilitate integration of biological data sources. The database community has focused on transforming relational and HTML data to XML since the inception of XML. Finally, the machine learning community has carried out research on learning extraction rules which occurred in mainly two contexts: creating wrappers for information agents and developing general purpose information extraction systems for natural language text. We review some of these technologies here.

Several systems have been designed for domain specific integration and warehousing of biological data [17]. However, to the best of our knowledge, none of these system transform flat files to XML using machine learning techniques. IBM's Discovery Link [10] is an SQL-based heterogeneous database integration system. It has most of the components required for data integration solutions, but they come in the wrong flavor [17]. They transform flat file data to the flat relational model directly instead of nested structure like XML. That is, they do not store nested objects in natural way. By doing so it becomes impotence in the biomedical data integration arena [17]. OPM [5] is a well designed system for the purpose of biomedical data integration, except for (1) a problem in performance due to data fragmentation as it unwisely maps all data to the third normal form, and (2) the lack of a simple data exchange format as they do not transform flat files into hierarchical XML, and (3) the need of a global schema. The Kleisli [6] system transforms and integrates heterogeneous data sources using a complex object data model and CPL, a powerful query language inspired by work in functional programming languages. In Kleisli the flat files are transformed to a nested data exchange format.

Machine learning approaches rely on learning from examples and counterexamples of a large number of training data. Stalker [15] specialises general SkipTo sequence patterns based on labelled HTML pages. An approach to maximize specific patterns is introduced by Davulcu et al. [7]. Other examples include Softmealy [11] that uses a wrapper induction algorithm that generates extraction rules expressed as finite transducers. NoDoSe [2] extracts information from plain string sources and provides a user interface for example labelling. It has restricted capabilities to deal with HTML. Kushmerick et al. [12] create robust wrappers based on predefined extractors; their visual support tool WIEN receives a set of training pages, where the user can label relevant information and the system tries to learn a wrapper.

Most of the above approaches deal with HTML pages instead of flat files. We believe that HTML pages structure are much more regular in nature compared to flat files due to the existence of tags. Specifically, in contrast to the approaches in [11,12] a key feature of Bio2X is that it is able to efficiently generate XML documents accurately from a small number of training data set: it rarely requires more than 12 examples. The ability to generalize from such a small number of examples has a two-fold explanation. First, most of the bio-elements in flat files are generated based on a fixed template that may have only a few variations. Only comment segment has free-text data. As Bio2X tries to learn landmarks that are part of this template, it follows that for templates with little or no variations a handful of examples usually will be sufficient to induce reliable landmarks. Moreover, WIEN can be seen as a non-disjunctive rules whereas Bio2X allows us to specify disjunctive rules. Similar to Stalker [15], we take a hierarchical approach to extract the XML elements/attributes from the flat files by decomposing the problem into a series of simple ones. However, Stalker supports a graphical user interface that allows a user to mark up several pages on a site, and the system then generates a set of extraction rules that extract the required information. We believe that such GUI can easily be build on top of Bio2X in the future.

In Tsimmis [9], the extraction process is based on a procedural program which skips to the required information, allows temporary storages, split and case statements, and to follow links. However, the wrapper output has to obey the document structure. In Araneus [3], a user can create relational views from web pages by computationally fast and advanced text extracting and restructuring formalisms, in particular using procedural "Cut and Paste" exception handling inside regular grammars. XWrap [14] uses a procedural rule system and provides limited expressive power for pattern definition. It is not possible to describe pattern disjunctions. These systems focus on transforming HTML data which has regular structure. Also, unlike our approach, these systems do not use machine learning technique to generate XML data.

7. Conclusions and future work

Data integration of geographically dispersed, heterogeneous, complex biological databases is a key research area. One of the key features of a successful data integration system is to have a simple self-describing data exchange formats [13]. However, many of the biological databases provide data in flat files which are poor data exchange format compared to XML. In this paper, we presented Bio2X system that transform flat file data into highly hierarchical XML data. A single transformer is implemented to transform the biological data from various data sources by applying the corresponding rule base. A rule specifies the way of extracting data, and the rule grammar is consistently designed to be suitable for each database. The rule base has also been refined with a machine-learning approach in order to improve the correctness of transformation. A working prototype of Bio2X has been implemented using Java. Preliminary results on representative flat files using the current Bio2X prototype show a good performance.

As part of future work, we would like to support a visual graphical user interface that allows a user to mark up several flat files easily and interactively in order to generate all extraction rules automatically.

References

[1] F. Achard, XML, bioinformatics, and data integration, Bioinformatics 17 (2001) 115–125.
[2] B. Adelberg, NoDoSE—a tool for semiautomatically extracting semi-structured data from text documents, in: Proc. SIGMOD, 1998.

[3] P. Atzeni, G. Mecca, Cut and paste, in: Proc. Princip. Database Syst. (PODS), 1997.
[4] C. Cardie, R. Mooney, ACL tutorial: symbolic machine learning for natural language processing, Available from <http://www.cs.cornell.edu/Info/People/cardie/tutorial/tutorial.html>, 1998.
[5] I.A. Chen, V.M. Markowitz, An overview of the object-protocol model and OPM data management tools, Inform. Syst. (1995).
[6] S.Y. Chung, L. Wong, Kleisli: a new tool for data integration in biology, Trends Biotechnol. 17 (9) (1999) 351–355.
[7] H. Davulcu, G. Yang, M. Kifer, I.V. Ramakrishnan, Computational aspects of resilient data extraction from semistructured sources, in: Proc. PODS, 2000.
[8] T. Etzold, A. Ulyanov, P. Argos, SRS: information retrieval system for molecular biology data banks, Meth. Enzymol. 266 (1996) 114–128.
[9] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, A. Crespo, Extracting semistructured information from the web, in: Proc. Workshop Mang. Semistruct. Data, 1997.
[10] L. Hass, P.M. Schwartz, P. Kodali, J. Rice, et al., Discovery link: a system for integrating life sciences data, in: Proc. IEEE Symp. Bio-Inform. Biomed. Eng. (BIBE 2000), Arlington, Virginia, 2000.
[11] C.N. Hsu, M.T. Dung, Generating finitestate transducers for semistructured data extraction from the web, in: Inform. Syst., Elsevier Science, 1998.
[12] N. Kushmerick, D. Weld, R. Doorenbos, Wrapper induction for information extraction, in: Proc. IJCAI, 1997.
[13] J. Li, S.K. Ng, L. Wong, Bioinformatics adventures in database research, in: Proc. 9th Int. Conf. Database Theory, Siena, Italy, January 2003, pp. 31–46.
[14] L. Liu, C. Pu, W. Han, XWrap: an extensible wrapper construction system for internet information, in: Proc. ICDE, 2000.
[15] I. Muslea, S. Minton, C. Knoblock, A hierarchical approach to wrapper induction, in: Proc. 3rd Int. Conf. Autonomous Agents, 1999.
[16] J. Shanmugasundaram, K. Tufte, C. Zhang, et al., Relational databases for querying XML documents: limitations and opportunities, in: Proc. 25th Int. Conf. Very Large Databases (VLDB 1999), Edinburgh, Scotland, 1999, pp. 302–314.
[17] L. Wong, Technologies for integrating biological data, Briefings Bioinform. 3 (4) (2002) 389–404.

Song Yang received her Bachelor's degree in Computer Engineering from Nanyang Technological University, Singapore in 2003. She is currently working as research engineer in Centre of High Performance Embedded Systems (CHIPES), Singapore.

Sourav S. Bhowmick received his Ph.D. in Computer Engineering from Nanyang Technological University, Singapore in 2001. He is an Assistant Professor of the School of Computer Engineering at the Nanyang Technological University. His current research interests include XML data management, mobile data management, biological data management, web warehousing and web mining. He has published more than 60 papers in major international database conferences and journals such as VLDB, ICDE, CIKM, ICDCS, ER, IEEE TKDE, ACM CS, DKE and DEXA. He is serving as PC member of various database conferences and workshops and reviewer for various database journals. He is the founding program chair for International Workshop on Biological Data Management (BIDM) held in conjunction with DEXA. He is a member of the ACM and IEEE Computer Society.

Sanjay Madria received his Ph.D. in Computer Science from Indian Institute of Technology, Delhi, India in 1995. He is an Assistant Professor of the Department of Computer Science at the University of Missouri-Rolla, USA. Earlier he was Visiting Assistant Professor in the Department of Computer Science, Purdue University, West Lafayette, USA. He has published more than 70 Journal and conference papers in the areas of web warehousing, mobile databases, data warehousing, nested transaction management and performance issues. He has chaired international conferences and workshops, organized tutorials, and has actively served in the program committees of numerous international conferences and has been reviewer for many journals. He participated as panelist in National Science Foundation and Swedish Research Council. He is an IEEE senior member and ACM member.