DBLP XML Requests

Appendix to the paper "DBLP — Some Lessons Learned" (June 17, 2009)

Michael Ley
Universität Trier, Informatik
D–54286 Trier
Germany
ley@uni-trier.de

ABSTRACT

If your software needs only in a few facts from DBLP, downloading the entire `dblp.xml` file may be a too costly burden. The web pages are intended for humans, wrappers are always exposed to the risk of formatting changes. In this appendix we describe a very basic API for DBLP. Our example application for the API is a simple crawler which finds the shortest path between two DBLP authors in the coauthor graph. In addition the appendix lists code to map person names to DBLP URLs.

DBLP Records

If you know the key of a DBLP record, you may retrieve the record from the URL

`http://dblp.uni-trier.de/rec/bibtex/`*key*`.xml`

e.g. if you replace *key* by `journals/acta/BayerM72`, you will get the bibliographic record of the famous B-tree paper:

```
<?xml version="1.0"?>
<dblp>
  <article key="journals/acta/BayerM72"
           mdate="2003-11-25">
    <author>Rudolf Bayer</author>
    <author>Edward M. McCreight</author>
    <title>Organization and Maintenance
      of Large Ordered Indices</title>
    ...
  </article>
</dblp>
```

Obviously this looks like a version of the huge `dblp.xml` file which has been shrunken to just one record. But there is an important difference: The header does not refer to an DTD and we do not use symbolic entities. All non-ASCII characters are encoded by numeric entities, in the header the encoding intentionally has not been specified. This pure-ASCII XML document without symbolic entities may be parsed very fast by a lightweight parser. The same encoding is used by the other services described in the sequel of this section.

Person Search

On `http://dblp.uni-trier.de/` DBLP provides a primitive form to search for persons inside the bibliography. If you type in `Schek`, you will get an answer on page

`http://dblp.uni-trier.de/search/author?author=Schek`

This page is an HTML document. Modify the URL by inserting an 'x' after the question mark:

`.../search/author?xauthor=Schek`

and you will get an XML version of the answer page:

```
<?xml version="1.0"?>
<authors>
  <author urlpt="s/Schek:Hans=J=ouml=rg">
    Hans-J&#246;rg Schek</author>
  <author urlpt="s/Schekelmann:Andr=eacute=">
    Andr&#233; Schekelmann</author>
</authors>
```

In the DBLP person search a query is interpreted as a set of prefixes of name parts. If you enter a few words, you get the names which include these words as prefixes of some name parts. The query string and the names stored in DBLP are broken in parts. The delimiters of this tokenizing are spaces and punctuation marks. The punctuation marks are not relevant for the matching. `Ar-b-c.` produces the same result as `Ar b c`. The matching is not case-sensitive. The order of the query words does not matter, i.e. the queries `Petra M A` and `M Petra A` are equivalent. For words which end with a $-sign, only exact matches of this word are shown. Try the queries `xi li`, `xi$li xi li$`, and `xi$li$` (`xi$ li` and `xi$li` are equivalent).

As long as you restrict your query to ASCII (< 128) the search engine matches 'diacritic insensitive', i.e. the query `moller` matches `moller`, `möller`, `møller`, `móller`, etc. As soon as your query contains any diacritic mark, the matching becomes exact for diacritics. Now `René` matches `René` but not `Rene` or `Renè`.

The preferred encoding to transmit the query is UTF-8. As soon as the query contains a byte sequence which is illegal in UTF-8, the incoming byte sequence is interpreted as Latin-1. Additionally, the search engine understands symbolic entities listed in `dblp.dtd`. The author search accepts queries using the GET or the POST method. The answer

length has a hard limit: If the query produces more than 1000 hits, the result is truncated.

In the XML version the answer is enclosed in an `authors` element. For each hit, it contains an `author` element with the matching person name. For convenience we have added the attribute `urlpt` to the `author` elements. This attribute contains the essential part of the name-to-URL mapping, by this for many applications it is no longer necessary to implement the `make_file_name` function described in the appendix. To get the URL of a person page, a *DBLPbURL*, *indices/a-tree/*, the `urlpt` value, and the `.html` suffix have to be concatenated.

If this simple search engine for persons is not sufficient for your purposes, you should implement your own one. The text file

`http://dblp.uni-trier.de/db/indices/AUTHORS`

lists all DBLP person names except homonyms (see next subsection). The encoding is pure ASCII, Latin-1 characters are represented by symbolic entities, each line contains one name.

Publications of a Person

The first part of a person page lists all known publications this person is involved in. This information is available in XML, too. Requests to URLs of the form

`http://dblp.uni-trier.de/rec/pers/`*urlpt*`/xk`

result in XML answers like (for *urlpt* = `Wong:Curtis`):

```
<?xml version="1.0"?>
<dblpperson name="Curtis Wong">
  <dblpkey>journals/sigmod/Wong08</dblpkey>
  <dblpkey>conf/chi/HuynhDBW05</dblpkey>
  ...
</dblpperson>
```

In the enclosing `dblpperson` element the attribute `name` specifies the person's name. The keys of the bibliographic records are contained in `dblpkey` elements. All records this person is involved in either as author or in the role of an editor are enumerated. If a "person record" has been created for this person, it's key is quoted ahead the regular bibliographic records. The entry is marked by an attribute:

`<dblpkey type="person record">homepages/...`

Additional information is provided if homonyms are known:

```
<?xml version="1.0"?>
<dblpperson name="Michael Meier">
  <homonym>m/Meier:Michael</homonym>
  <homonym>m/Meier_0004:Michael</homonym>
  <homonym>m/Meier_0003:Michael</homonym>
  <dblpkey type="person record">
    homepages/m/MichaelMeier2</dblpkey>
  <dblpkey>conf/edbt/LausenMS08</dblpkey>
  <dblpkey>journals/corr/abs-0812-3788</dblpkey>
</dblpperson>
```

Our example was requested from

`.../rec/pers/m/Meier_0002:Michael/xk`

i.e. these are the keys for Michael Meier with the ID suffix `0002`. The `name` does not contain the ID suffix. For the other Michael Meiers the urlpts are enumerated in `homonym` elements in front of the key list. The person search only returns the base name without ID suffix. For the query

`.../search/author?xauthor=Michael+Meier`

DBLP returns only

```
<?xml version="1.0"?>
<authors>
  <author urlpt="m/Meier:Michael">
    Michael Meier</author>
</authors>
```

but the `homonym` elements shown above make it easy to deal with ambiguous person names.

Coauthors

If you load all bibliographic records of a person, it is a trivial step to get the set of coauthors for this person. But loading all bibliographic records may be expensive. The URLs

`http://dblp.uni-trier.de/rec/pers/`*urlpt*`/xc`

give direct access to the coauthor lists. For example

`.../rec/pers/h/Halevy:Alon_Y=/xc`

returns the XML document

```
<?xml version="1.0"?>
<coauthors author="Alon Y. Halevy">
  <author urlpt="a/Abiteboul:Serge"
    count="3">Serge Abiteboul</author>
  ...
  <author urlpt="z/Zhang:Yang"
    count="2">Yang Zhang</author>
</coauthors>
```

The `count` attribute specifies the number of shared publications of the two persons.

In a future version we plan to extend the XMI API for DBLP. Access to individual BHT files is not possible yet, but it may be beneficial for applications which are interested in the tables of contents pages. Indices for DOIs, ISBN and other fields may be useful.

Shortest Path Algorithm

The java program[1] documented in this section demonstrates the use of one of the DBLP XML services described above. It computes the shortest path between two DBLP authors in the coauthor graph. The software works like a little web crawler, it loads the information incrementally from the DBLP server.

The program is structured in two classes: The shortest path algorithm is a variant of the classic breasth-first algorithm, it is shown in figure 1. The class `Person` shown in figure 2 hides all DBLP specific implementation details from the algorithms itself.

The interaction beween both classes is very simple: If you have a `Person` object, you may ask it for its neighbors by applying the method `getCoauthors`, which returns an array of persons. You can attach a label to each person. A label is an integer. `setLabel` creates a label, `hasLabel` tests

[1]An expanded version of this software is available from `http://dblp.uni-trier.de/xml/docu/`

```
import java.util.Collection;
import java.util.HashSet;
import java.util.Iterator;

public class CoauthorPath {
    private Person path[];

    public CoauthorPath(Person p1, Person p2)
        { shortestPath(p1,p2); }

    public Person[] getPath() { return path; }

    private void tracing(int position) {
        Person pNow, pNext;
        int direction, i, label;

        label = path[position].getLabel();
        direction = Integer.signum(label);
        label -= direction;
        while (label != 0) {
            pNow = path[position];
            Person ca[] = pNow.getCoauthors();
            for (i=0; i<ca.length; i++) {
                pNext = ca[i];
                if (!pNext.hasLabel())
                    continue;
                if (pNext.getLabel() == label) {
                    position -= direction;
                    label -= direction;
                    path[position] = pNext;
                    break;
    }   }   }   }

    private void shortestPath(Person p1, Person p2) {
        Collection<Person>  h,
          now1 = new HashSet<Person>(),
          now2 = new HashSet<Person>(),
          next = new HashSet<Person>();
        int direction, label, n;

        Person.resetAllLabels();
        if (p1 == null || p2 == null)
            return;
        if (p1 == p2) {
            p1.setLabel(1);
            path = new Person[1];
            path[0] = p1;
            return;
        }
        p1.setLabel( 1); now1.add(p1);
        p2.setLabel(-1); now2.add(p2);

        while (true) {
            if (now1.isEmpty() || now2.isEmpty())
                return;

            if (now2.size() < now1.size()) {
                h = now1; now1 = now2; now2 = h;
            }

            Iterator<Person> nowI = now1.iterator();
            while (nowI.hasNext()) {
                Person pnow = nowI.next();
                label = pnow.getLabel();
                direction = Integer.signum(label);
                Person neighbours[] = pnow.getCoauthors();
                int i;
                for (i=0; i<neighbours.length; i++) {
                    Person px = neighbours[i];
                    if (px.hasLabel()) {
                        if (Integer.signum(px.getLabel())
                                ==direction)
                            continue;
                        if (direction < 0) {
                            Person ph;
                            ph = px; px = pnow; pnow = ph;
                        }
                        // pnow has a positive label,
                        // px a negative
                        n = pnow.getLabel() - px.getLabel();
                        path = new Person[n];
                        path[pnow.getLabel()-1] = pnow;
                        path[n+px.getLabel()] = px;
                        tracing(pnow.getLabel()-1);
                        tracing(n+px.getLabel());
                        return;
                    }
                    px.setLabel(label+direction);
                    next.add(px);
                }
            }
            now1.clear(); h = now1; now1 = next; next = h;
        }
    }
}
```

Figure 1: A shortest path algorithm

```
import ...;

public class Person {
    private static Map<String, Person> personMap =
            new HashMap<String, Person>();
    private String name;
    private String urlpt;

    private Person(String name, String urlpt) {
        this.name = name;
        this.urlpt = urlpt;
        personMap.put(name, this);
        coauthorsLoaded = false;
        labelvalid = false;
    }

    static public Person create(String name, String urlpt) {
        Person p;
        p = searchPerson(name);
        if (p == null)
            p = new Person(name, urlpt);
        return p;
    }

    static public Person searchPerson(String name) {
        return personMap.get(name);
    }

    private boolean coauthorsLoaded;
    private Person coauthors[];

    static private SAXParser coauthorParser;
    static private CAConfigHandler coauthorHandler;
    static private List<Person> plist
            = new ArrayList<Person>();

    static private class CAConfigHandler
                    extends DefaultHandler {
        private String Value, urlpt;
        private boolean insideAuthor;

        public void startElement(String namespaceURI,
                String localName, String rawName,
                Attributes atts) throws SAXException {
            if (insideAuthor = rawName.equals("author")) {
                Value = "";
                urlpt = atts.getValue("urlpt");
        }   }

        public void endElement(String namespaceURI,
                    String localName, String rawName)
                            throws SAXException {
            if (rawName.equals("author") &&
                    Value.length() > 0) {
                plist.add(create(Value, urlpt));
        }   }

        public void characters(char[] ch, int start, int length)
                throws SAXException {
            if (insideAuthor)
                Value += new String(ch, start, length);
        }

        public void warning(SAXParseException e)
                throws SAXException { ... }
        public void error(SAXParseException e)
                throws SAXException { ... }
        public void fatalError(SAXParseException e)
                throws SAXException { ... }
    }

    static {
        try { coauthorParser = SAXParserFactory.
                newInstance().newSAXParser();
            coauthorHandler = new CAConfigHandler();
            coauthorParser.getXMLReader().setFeature(
             "http://xml.org/sax/features/validation",
             false);
        } catch (ParserConfigurationException e) { ...
        } catch (SAXException e) { ... }
    }

    private void loadCoauthors() {
        if (coauthorsLoaded)
            return;
        plist.clear();
        try { URL u = new URL(
            "http://dblp.uni-trier.de/rec/pers/"
             +urlpt+"/xc");
            coauthorParser.parse(u.openStream(),
                            coauthorHandler);
        } catch (IOException e) { ...
        } catch (SAXException e) { ... }
        coauthors = new Person[plist.size()];
        coauthors = plist.toArray(coauthors);
        coauthorsLoaded = true;
    }

    public Person[] getCoauthors() {
        if (!coauthorsLoaded) { loadCoauthors(); }
        return coauthors;
    }

    private int label;
    private boolean labelvalid;

    public int getLabel() { return (!labelvalid)?0:label; }
    public void resetLabel() { labelvalid = false; }
    public boolean hasLabel() { return labelvalid; }

    public void setLabel(int label) {
        this.label = label;
        labelvalid = true;
    }

    static public void resetAllLabels() {
        Iterator<Person> i = personMap.values().iterator();
        while (i.hasNext()) {
            Person p = i.next();
            p.labelvalid = false;
            p.label = 0;
    }   }

    public String toString() { return name; }
}
```

Figure 2: The Class "Person"

if a label exists, and `getLabel` reads a label of a person. The class method `resetAllLabels` deletes all labels from persons. It is obvious that you may generalize this to a simple interface to access any undirected graph (without weights) from your shortest path algorithm.

To run the algorithm, you have to create two persons and to construct a `CoauthorPath` object:

```
Person p1, p2;
p1 = Person.create("Jim Gray", "g/Gray:Jim");
p2 = Person.create(...);
CoauthorPath cp = new CoauthorPath(p1,p2);
Person path[] = cp.getPath();
```

After this you may print out the path.

Next we look at the class `Person` more closely. In the code snipplet above it is noticeable that we use the class method `create` to produce new `Person` objects. The class does not have a public constructor because it caches all objects in a class level `Map`. The `create` method first tests if there is already an object for the specified name in `personMap`. A new object is only created, if this test fails.

After creation a `Person` object only contains the person's name and the `urlpt`. The list of coauthors is only loaded on demand[2]. The boolean field `coauthorsLoaded` contains the required state information. If `getCoauthors` is called for the first time, `loadCoauthors` is activated to fetch the information from DBLP.

We use a SAX parser to read the XML file. Because the creation of a SAX parser object is an expensive task, the parser is reused. It is created in the static initialization block just above `loadCoauthors` at class load time. Additionally the `coauthorHandler` field is initialised. In `loadCoauthors` an `URL` object is constructed to provide an `InputStream` for the parser. During the parsing process the SAX parser calls the methods `startElement`, `endElement`, and `characters` of the local `CAConfigHandler`. We are only interested in `author` elements. The boolean `insideAuthor` is set `true` as soon as we recognize an opening `author` tag. `characters` collects the element contents in `Value`. In the method `endElement` a `Person` object is created. The coauthors are temporarily stored in `plist`. This `List` is converted into the final `coauthors` array after parsing has been completed.

To make the shortest path algorithm practicable over a slow internet connection, we have to minimize the number co-author lists to be loaded. Two known optimizations of the breadth-first algorithm proved to be essential for searches inside the huge connected component of the DBLP coauthor graph: (1) The search has to start from both persons, and (2) the algorithm should prefer the side with the lower number of persons to be visited next. The method `CoauthorPath.shortestPath` is a straightforward implementation of these ideas.

The algorithm labels persons `p1` with 1 and `p2` with -1. The direct neighbors of `p1` are set 2, the direct neighbors of `p2` are set -2 etc. The variables `now1` and `now2` contain the sets of persons who form the outer waves of the labeling processes. The main loop flips the side where to advance next depending on the size of these sets. During an expansion step the still unvisited persons are collected in the set `next`.

[2]In the online version of the class the same mechanism is implemented for the list of publications of a person.

If `now1` or `now2` become empty, the persons are not connected. If the two waves hit each other, the method `tracing` is called to collect pathes from the meeting point to the starting points.

make_file_name

This primitive C program shows the function which calculates the URL of a DBLP author page from a name. For a production version, you should add some tests to prevent buffer overflow attacks.

```
#include <stdlib.h>
typedef char line[10000];
line name_array, file_name_array;

char map_char(char x) {
  if (isalnum(x)) return x;
  return (x == ' ') ? '_' : '=';
}
int is_hidden_suffix(char *x) {
  if (x == NULL)   return 0;
  if (!isdigit(x[0]) || !isdigit(x[1]) ||
      !isdigit(x[2]) || !isdigit(x[3])) return 0;
  return x[4] == '\0';
}

char *make_file_name(char *name) {
  int  i = 0;
  char c, *lname, *fname, *help;

  strcpy(name_array,name);
  lname = strrchr(name_array,' ');
  if (lname) {
    fname = name_array;
    *lname++ = '\0';
    if (strcmp(lname,"Jr.")==0 ||
        strcmp(lname,"II") ==0 ||
        strcmp(lname,"III")==0 ||
        strcmp(lname,"IV")==0  ||
        is_hidden_suffix(lname)) {
      help = strrchr(fname,' ');
      if (help) {
        --lname;  *lname = ' ';
        *help = '\0';
        lname = help+1;
      }
    }
  } else {
    fname = strrchr(name_array, '\0');
    lname = name_array;
  }
  if (lname)
    while (c = *lname++)
      file_name_array[i++] = map_char(c);
  file_name_array[i++] = ':';
  if (fname)
    while (c = *fname++)
      file_name_array[i++] = map_char(c);
  file_name_array[i] = '\0';
  return file_name_array;
}

void print_keys_url(char *name) {
```

```
  char *s = make_file_name(name);
  printf("http://dblp.uni-trier.de"
    "/rec/pers/%c/%s/xk\n", tolower(*s),s);
}

main() {
    print_keys_url("Lars M&ouml;nch");
    print_keys_url("Chris van den Bos");
    print_keys_url("Luqi");
}
```