Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

ETLs for importing UniProtKB, HGNC, and Reactome into a bioinformatics graph database

A. Messina

Rapporto Tecnico N.:
RT-ICAR-PA-16-03 **Febbraio 2016**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sede di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sede di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

ETLs for importing UniProtKB, HGNC, and Reactome into a bioinformatics graph database

A. Messina[1]

Rapporto Tecnico N.:
RT-ICAR-PA-16-03 **Febbraio 2016**

---

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Palermo, Via Ugo La Malfa n. 153, 90146 Palermo.

Index

1 INTRODUCTION 4

2 UNIPROTKB 5

2.1 Introduction 5

2.2 Data source 5

2.3 ETL 6

3 HGNC 8

3.1 Introduction 8

3.2 Data source 8

3.3 ETL 8

4 REACTOME 12

4.1 Introduction 12

4.2 Data source 12

4.3 ETL 12

5 REFERENCES 14

1 Introduction

This work is the second of a series of technical report documenting the performed activities to build a big bioinformatics database.

Details on the target platform, the general Extraction-Transform-Load (ETL) template, and the already imported bioinformatics database can be found in [1].

In this technical report other source bioinformatics databases are considered, among with the related ETLs used for the import process.

2 UniProtKB

2.1 Introduction

The UniProt Knowledgebase (UniProtKB) [2] is the largest public collection of annotated functional information on proteins and it is updated every four weeks. It stores both computationally analyzed and manually annotated records, including classifications, cross-references and quality indications available to scientific researchers.

UniProtKB is actually composed of two sections: UniProtKB/Swiss-Prot and UniProtKB/TrEMBL. UniProtKB/Swiss-Prot is the reviewed section of the UniProt Knowledgebase. The TrEMBL section of UniProtKB was introduced in 1996 in response to the increased dataflow resulting from genome projects. It was already recognized at that time that the traditional time- and labour-intensive manual curation process which is the hallmark of Swiss-Prot could not be broadened to encompass all available protein sequences. UniProtKB/TrEMBL contains high quality computationally analyzed records that are enriched with automatic annotation and classification. These UniProtKB/TrEMBL unreviewed entries are kept separated from the UniProtKB/Swiss-Prot manually reviewed entries so that the high quality data of the latter is not diluted in any way. Automatic processing of the data enables the records to be made available to the public quickly.

2.2 Data source

UniProtKB is available for download from the URL ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete/ and it is released in three different file formats: XML, fasta, and text.

Thanks to the availability of well documented and powerful XML API in the Java language, in this work the XML version of UniProKB/Swiss-Prot was utilized. It is accompanied by an XML schema file, which describes the structure of the XML document.

The XML Schema language is also referred to as XML Schema Definition (XSD). The purpose of an XML Schema is to define the legal building blocks of an XML document:

- the elements and attributes that can appear in a document
- the number of (and order of) child elements

- data types for elements and attributes
- default and fixed values for elements and attributes

2.3 ETL

In order to get a Java representation of the XML schema cited above, it is necessary to map the elements of the schema to members of a (or more) java class. This transformation can be done using the data binder JAXB [3].

JAXB generates classes and groups them in Java packages. A package consists of a Java class name and an *ObjectFactory* class. The latter is a factory that is used to return instances of a bound Java class.

After data bindings exist, the JAXB binding runtime API can be used to convert XML instance documents to and from Java objects. Data stored in an XML document is accessible without the need to understand the data structure. JAXB annotated classes and artifacts contains all the information that the JAXB runtime API needs to process XML instance documents. The JAXB runtime API enables marshaling of JAXB objects to XML and unmarshaling the XML document back to JAXB class instances.

In this case, the ETL reads the XML source document by using the Streaming API for XML (StAX) [4]. It provides the interface *XMLStreamReader*, which gives a low-level but very efficient cursor-like API for reading XML documents. When using it we iterate over various events in XML document and extract information about these events. Once we are done with the current event, we move to the next one and continue. The events can be for example the start of element, the end of element or characters data.

```
…

JAXBContext jc = JAXBContext.newInstance(Entry.class);
Unmarshaller unmarshaller = jc.createUnmarshaller();

while (xsr.nextTag() == XMLStreamConstants.START_ELEMENT) {
    Entry entry = (Entry) unmarshaller.unmarshal(xsr);

    OrganismType organism = entry.getOrganism();
    String organismTaxonomyId = ((organism != null) &&
                  (!organism.getDbReference().isEmpty())) ?
                  organism.getDbReference().get(0).getId() : "";

    if (organismTaxonomyId.equals("9606")) {
        if (entry.getAccession().isEmpty())
            continue;

        ProteinType prot = entry.getProtein();

        //String accession = entry.getAccession().get(0);
```

```
        String name = entry.getName().get(0);
        String fullName = ((prot.getRecommendedName() != null) &&
           (prot.getRecommendedName().getFullName() != null)) ?
           prot.getRecommendedName().getFullName().getValue() : "";
        String alternativeName =
           ((!prot.getAlternativeName().isEmpty()) &&
        (prot.getAlternativeName().get(0).getFullName() != null)) ?
         prot.getAlternativeName().get(0).getFullName().getValue()
         : "";

        SequenceType seq = entry.getSequence();
        String sequence = seq.getValue();
        int sequenceLenght = seq.getLength();
        int sequenceMass = seq.getMass();

        Vertex protein = graph.addVertex("class:protein",
                    "name", name,
                    "fullName", fullName,
                    "alternativeName", alternativeName,
                    "sequence", sequence,
                    "sequenceLenght", sequenceLenght,
                    "sequenceMass", sequenceMass
                    );

        for (String accessionName : entry.getAccession()) {
             Vertex accession =
                       graph.addVertex("class:proteinAccession",
                          "name", accessionName
                          );
             accession.addEdge("refersTo", protein);
        }
}

…
```

3 HGNC

3.1 Introduction

The HUGO Gene Nomenclature Committee (HGNC) [3] is the authority responsible for the gene nomenclatures (also known as gene symbols) for the human species. This authority also provides the HGNC database, that contains, for each gene symbol, a list of synonyms and a list of corresponding entries in the most popular gene databases (e.g. Refseq, Entrez gene).

3.2 Data source

The web page at URL http://www.genenames.org/cgi-bin/statistics shows a table that contains the number of genes associated to locus groups and types. Within the table there also are links to download the data for each locus group or type. The first source data file used in this work has been downloaded from the URL ftp://ftp.ebi.ac.uk/pub/databases/genenames/new/tsv/locus_groups/protein-coding_gene.txt and it has been used to load all the symbol synonyms of a given gene and to create associations of type coding between genes and proteins. Another data file, found at the URL ftp://ftp.ebi.ac.uk/pub/databases/genenames/new/tsv/locus_groups/non-coding_RNA.txt, has been instead used only to load other symbol synonyms for genes.

3.3 ETL

The type of source files is tab-delimited separated values, but the columns number is not constant line by line. This implies many controls on the length of the *datavalue* array, in order to prevent runtime errors caused by array indexes out of bounds.

The columns containing genes synonyms have been read from both source files. From the *protein-coding_gene.txt* the coded protein is read, too.

```
…

while ((line = reader.readLine()) != null) {
    String datavalue[] = line.split("\t");

    String hgncId = datavalue[0];
    /*
    String name = datavalue[2];
```

```
    String locusGroup = datavalue[3];
    String locusType = datavalue[4];
    String status = datavalue[5];
    String location = datavalue[6];
    String locationSortable = datavalue[7];
    String aliasSymbol = datavalue[8];
    String aliasName = datavalue[9];
    String prevSymbol = datavalue[10];
    String prevName = datavalue[11];
    String geneFamily = datavalue[12];
    String geneFamilyId = datavalue[13];
    String dateApprovedReserved = datavalue[14];
    String dateSymbolChanged = datavalue[15];
    String dateNameChanged = datavalue[16];
    String dateModified = datavalue[17];
    */
    String entrezId = datavalue[18];
    String ensemblGeneId = datavalue[19];
    /*
    String vegaId = datavalue[20];
    String ucscId = datavalue[21];
    String ena = datavalue[22];
    */
    String refseqAccession = datavalue[23];
    if (datavalue.length < 25)
        continue;
    //String ccdsId = datavalue[24];
    if (datavalue.length < 26)
        continue;
    String uniprotIds = datavalue[25];

    /*
    if (datavalue.length < 27)
        continue;
    String pubmedId = datavalue[26];
    if (datavalue.length < 28)
        continue;
    String mgdId = datavalue[27];
    if (datavalue.length < 29)
        continue;
    String rgdId = datavalue[28];
    if (datavalue.length < 30)
        continue;
    String lsdb = datavalue[29];
    if (datavalue.length < 31)
        continue;
    String cosmic = datavalue[30];
    if (datavalue.length < 32)
        continue;
    String omimId = datavalue[31];
    if (datavalue.length < 33)
        continue;
    String mirbase = datavalue[32];
    String homeodb = datavalue[33];
    if (datavalue.length < 35)
        continue;
    String snornabase = datavalue[34];
    String bioparadigmsSlc = datavalue[35];
    if (datavalue.length < 37)
        continue;
    String orphanet = datavalue[36];
    if (datavalue.length < 38)
        continue;
    String pseudogene = datavalue[37];
    if (datavalue.length < 39)
        continue;
    String hordeId = datavalue[38];
    if (datavalue.length < 40)
```

```
        continue;
    String merops = datavalue[39];
    if (datavalue.length < 41)
        continue;
    String imgt = datavalue[40];
    String iuphar = datavalue[41];
    if (datavalue.length < 43)
        continue;
    String kznfGeneCatalog = datavalue[42];
    if (datavalue.length < 44)
        continue;
    String mamitTrnadb = datavalue[43];
    String cd = datavalue[44];
    if (datavalue.length < 46)
        continue;
    String lncrnadb = datavalue[45];
    String enzymeId = datavalue[46];
    if (datavalue.length < 48)
        continue;
    String intermediateFilamentDb = datavalue[47];
    */

    if (!entrezId.equals("")) {
        Vertex gene = null;

        Iterator<Vertex> it =
             graph.getVertices("gene.geneId", entrezId).iterator();

        if (it.hasNext())
             gene = it.next();

        if (gene != null) {
             Vertex ncbi = graph.addVertex("class:geneName",
                                       "symbol", entrezId
                                       );
             ncbi.addEdge("synonymOf", gene);

             String entrezSymbol = gene.getProperty("symbol");
             Vertex symbol = graph.addVertex("class:geneName",
                                       "symbol", entrezSymbol
                                       );
             symbol.addEdge("synonymOf", gene);

             if ((hgncId != null) && (!hgncId.equals(""))) {
                   Vertex hgnc = graph.addVertex("class:geneName",
                                       "symbol", hgncId
                                       );
                   hgnc.addEdge("synonymOf", gene);
             }


             if ((ensemblGeneId != null) &&
                (!ensemblGeneId.equals(""))) {
                   Vertex ensembl =
                       graph.addVertex("class:geneName",
                                       "symbol", ensemblGeneId
                                       );
                   ensembl.addEdge("synonymOf", gene);
             }

             if ((refseqAccession != null) &&
                (!refseqAccession.equals(""))) {
                   Vertex refseq =
                       graph.addVertex("class:geneName",
                                       "symbol", refseqAccession
                                       );
                   refseq.addEdge("synonymOf", gene);
             }
```

```
            if (!uniprotIds.equals("")) {
               if (uniprotIds.contains("|")) {
                   String uniprotId[] = uniprotIds.split("|");

                   for (int i=0; i<uniprotId.length; i++) {
                      it =
                        graph.getVertices("proteinAccession.name",
                                          uniprotId[i]).iterator();
                      if (it.hasNext()) {
                          Edge e = it.next().
                               getEdges(Direction.OUT,
                               "refersTo").iterator().next();
                           Vertex protein =
                               e.getVertex(Direction.IN);

                          graph.addEdge("class:coding", gene,
                                        protein, "coding");
                      }
                   }
               } else {
                   it = graph.getVertices("proteinAccession.name",
                               uniprotIds).iterator();
                   if (it.hasNext()) {
                       Edge e = it.next().getEdges(Direction.OUT,
                                 "refersTo").iterator().next();
                       Vertex protein = e.getVertex(Direction.IN);

                       graph.addEdge("class:coding", gene,
                                     protein, "coding");
                   }
               }
            }
         }
      }
}

…
```

4 Reactome

4.1 Introduction

Reactome [4] is a database containing validated metabolic pathways in human biology and computationally inferred pathways for 20 non-human species. Each pathway is annotated as a set of biological events, dealing with genes and proteins. In order to support the scientific community, it provides a mapping among these entities and the main source database identifiers, such as UniprotKB and miRBase [5].

4.2 Data source

The download section of Reactome site at the URL http://www.reactome.org/pages/download-data/ contains many downloadable resources, such as:

- Identifier mapping files, which link the source database identifier to the lowest level pathway diagram or subset of the pathway or to all levels of the pathway hierarchy;
- Pathway information, that is the complete list of pathways and the pathway hierarchy relationship;
- Interactions derived from Reactome pathways, such as the human protein-protein interaction pairs in tab-delimited format;
- MySQL dumps of Reactome database.

By executing some specifically custom SQL views, data of interest have been directly extracted from the SQL dumps of the database, which were previously imported into a dedicated MySQL server instance. Examples of such data are: pathway to disease relations, pathways summations, literature references.

4.3 ETL

The Reactome ETL is a bit complex compared to the other ones, because it also contains an extra data loading section, in order to load, in memory, the results of the queries on the views cited above.

Again, just the biological entities related to the human species are considered. For the pathways, it means just the pathways with the prefix "*R-HAS-*" are considered for the import.

```
…

        for (String id : pathwayName.keySet()) {
              String name = pathwayName.get(id);
              String disease = pathwayDisease.get(id);
              String summation = pathwaySummation.get(id);
              ArrayList<String> literatureReference =
pathwayLiteratureReference.get(id);
              ArrayList<String> summationLiteratureReference =
pathwaySummationLiteratureReference.get(id);

              Vertex p = graph.addVertex("class:pathway",
                           "id", id,
                           "name", name,
                           "disease", disease,
                           "summation", summation
                           );
       pathway.put(id, p);

              if (literatureReference != null)
                     for (String reference : literatureReference) {
                     Iterator<Vertex> it =
graph.getVertices("pubmed.id", reference).iterator();
                           Vertex citation = it.hasNext() ?
it.next() : graph.addVertex("class:pubmed", "id", reference);
                           p.addEdge("citedIn", citation);
                     }

              if (summationLiteratureReference != null)
                     for (String reference :
summationLiteratureReference) {
                     Iterator<Vertex> it =
graph.getVertices("pubmed.id", reference).iterator();
                           Vertex citation = it.hasNext() ?
it.next() : graph.addVertex("class:pubmed", "id", reference);
                           p.addEdge("citedIn", citation);
                     }
}

…

        while ((line = reader.readLine()) != null) {
              String datavalue[] = line.split("\t");
               String accessionId = datavalue[0];
              String pathwayId = datavalue[1];

              if (accessionId.startsWith("miR"))
                     accessionId = "MI" + accessionId.substring(3);

              Vertex m = null;
              Vertex p = pathway.get(pathwayId);

              Iterator<Vertex> it =
graph.getVertices("miRNA.accession", accessionId).iterator();
              if (it.hasNext())
                     m = it.next();

              if ((p != null) && (m != null)) {
                     m.addEdge("miRNA2pathway", p);

       }
}

…
```

5 References

[1] A. Messina, "ETLs for importing NCBI Entrez Gene, miRBase, mirCancer and microRNA into a bioinformatics graph database," Palermo, 2015.

[2] The UniProt Consortium, "UniProt: a hub for protein information," *Nucleic Acids Research,* vol. 43, no. D1, pp. D204-D212, 2015.

[3] Java Community Process, "JSR 222: Java Architecture for XML Binding (JAXB) 2.0," 2009. [Online]. Available: https://jcp.org/en/jsr/detail?id=222. [Accessed 04 02 2016].

[4] Java Community Process, "JSR 173: Streaming API for XML," 04 Mar 2014. [Online]. Available: https://jcp.org/en/jsr/detail?id=173. [Accessed 12 Feb 2016].

[5] K. A. Gray, B. Yates, R. L. Seal, M. W. Wright and E. A. Bruford, "Genenames.org: the HGNC resources in 2015," *Nucleic Acids Research,* vol. 43, no. D1, pp. D1079-D1085, 2015.

[6] D. Croft, A. F. Mundo, R. Haw, M. Milacic, J. Weiser, G. Wu, M. Caudy, P. Garapati, M. Gillespie, M. R. Kamdar, B. Jassal, S. Jupe, L. Matthews, B. May, S. Palatnik, K. Rothfels, V. Shamovsky, H. Song, M. Williams, E. Birney, H. Hermjakob, L. Stein and P. D'Eustachio, "The Reactome pathway knowledgebase," *Nucleic Acids Research,,* vol. 42, no. D1, pp. D474-7, 2014.

[7] A. Kozomara and S. Griffiths-Jones, "miRBase: integrating microRNA annotation and deep-sequencing data," *Nucleic acids research,* vol. 39, no. D1, pp. D152-7, 2011.