Accessing biological data in R with semantic web technologies

Egon L. Willighagen[1,2]

February 1, 2014

[1] Department of Bioinformatics - BiGCaT, NUTRIM, Maastricht University, Maastricht, The Netherlands
[2] Division of Molecular Toxicology, IMM, Karolinska Institutet, Stockholm, Sweden

*Corresponding author
Email: egon.willighagen@maastrichtuniversity.nl
Twitter: @egonwillighagen
ORCID: http://orcid.org/0000-0001-7542-0286

Manuscript type: research article

Keywords: semantic web, workflow

PeerJ PrePrints | http://dx.doi.org/10.7287/peerj.preprints.185v2 | CC-BY 3.0 Open Access | received: 2 Feb 2014, published: 2 Feb 2014

Abstract

Background. Semantic Web technologies are increasingly used in biological database systems. The improved expressiveness show advantages in tracking provenance and allowing knowledge to be more explicitly annotated. The list of semantic web standards needs a complementary set of tools to handle data in those formats to use them in bioinformatics workflows.
Methods. The approach proposed in this paper uses the Apache Jena library to create an environment where semantic web technologies can be use in the statistical environment R. The code is exposed as two R packages available from the Comprehensive R Archive Network (CRAN). The RJava library and a custom convenience class is used to bridge between R and the Jena library.
Results. We here present three examples showing how the Resource Description Framework (RDF) and SPARQL query standards can be employed in R. The first example takes input on BRCA1 SNPs from a BioMart and converts this into a RDF data set. The second example runs a query on an experimental remote SPARQL end point provided by Uniprot, and searches textual annotations of proteins encoded by the BRCA1 gene. The third example shows how the package can be used to handle RDF returned by OpenTox web services.
Discussion. The two provided library bring basic semantic web technologies to R. While only a subset of Apache Jena is currently exposed, it provides key methods to deal with RDF data and resources. The libraries are freely available from the CRAN under the Affero GNU Public License version 3: `http://cran.r-project.org/web/packages/rrdf/`.

1 Introduction

Semantic Web technologies are finding their way into biological databases [1, 2, 3, 4, 5]. The recent launch of the Resource Description Framework (RDF) services at the European Bioinformatics Institute is a good example of the impact it has on the field, despite the adoption taking place for longer already [6]. At the same time, most new databases do not yet use semantic web technologies, which may be partly caused by the lack of availability of tools to handle RDF data.

RDF allows the combination of a machine readable framework with the use of ontologies that allow sharing data and the meaning in the some data exchange format. This is enabled by a set of open specifications, such as the underlying RDF, RDF Schema, the Web Ontology Language, and serialization formats such as RDF/XML, N-TRIPLES, Notation3 (N3), and Turtle. Crucial, however, seems the specification of the SPARQL query language that allow extracting data from local or remote RDF triple stores [7]. In fact, federated SPARQL queries allow extracting data from multiple triple stores at the same time.

At the same time, R has established itself as a key tool for statistics, particularly in the biological sciences. Many packages have been developed for this domain and are available from CRAN and Bioconductor, allowing to combine various bioinformatics and data analysis approaches. These packages include, for example, biomaRt, which have the purpose of extracting information from a remote database.

PeerJ PrePrints | http://dx.doi.org/10.7287/peerj.preprints.185v2 | CC-BY 3.0 Open Access | received: 2 Feb 2014, published: 2 Feb 2014

We here introduce two packages, rrdflibs and rrdf, for the statistics environment R, based on the Apache Jena library and the RJava interface to Java [8, 9, 10]. Compared to the existing SPARQL package, this package adds general RDF data handling [11]. Using this package we show how data aggregation can be performed from remote SPARQL end points, and how local data in other formats can be converted into RDF to querying with SPARQL.

2 Materials & Methods

The rrdf packages provides RDF and SPARQL functionality by exposing that functionality from the Apache Jena library [9]. The rrdflibs package contains the Apache Jena libraries and nothing more. This package is close to 7 MiB large but does not change frequently. The rrdf package contains the R functions that wrap around the Jena functionality and convert data structures where needed. This package is about 112 kiB but changes more frequently. Using this approach, the Comprehensive R Archive Network (CRAN) server has the least amount of download throughput. The Jena functionality is accessible using the rJava library [10] which handles loading of the Java archives and provides methods to instantiate classes and call methods.

The source code of the package is available from GitHub at `http://github.com/egonw/rrdf/` and binary packages are available from CRAN at `http://cran.r-project.org/web/packages/rrdf/`. The first patch is from March 2011 while the most recent patches are from late 2013 when the rrdflibs package was updated for Apache Jena 2.11.

3 Results and Discussion

The rrdf package provides basic and less basic functionality for aggregation and analysis of RDF data. The package can be install and loaded with these commands:

```
install.packages(c("rrdf", "rrdflibs"))
library(rrdf)
```

3.1 Triple stores

To handle triples, we first need a triple store. At this moment only in-memory stores are supported, though Jena's TDB also provides on-disk triple stores; this puts limitations to the amount of data you can analyze in one study. The package supports two kinds of stores, one that has minimal ontology support, and one basically just handles triples. Both can be created with the *new.rdf* command:

```
ontStore = new.rdf()
store = new.rdf(ontology=FALSE)
```

When data is not read from a file or downloaded, but created from data in another format, such as an R matrix, individual triples can be added to a store. This works for both object properties and data properties. The former option links a subject to an object resource, while the latter links a subject to a literal value:

PeerJ PrePrints | http://dx.doi.org/10.7287/peerj.preprints.185v2 | CC-BY 3.0 Open Access | received: 2 Feb 2014, published: 2 Feb 2014

```
add.triple(store,
  subject="http://example.org/Subject",
  predicate="http://example.org/Predicate",
  object="http://example.org/Object"
)
add.data.triple(store,
  subject="http://example.org/Subject",
  predicate="http://example.org/Predicate",
  data="Literal value"
)
```

We can also take advantage from the fact that R assigns values to method parameters based on the order in which they are given. Therefore, we can simplify the above code to:

```
add.triple(store,
  "http://example.org/Subject",
  "http://example.org/Predicate",
  "http://example.org/Object"
)
add.data.triple(store,
  "http://example.org/Subject",
  "http://example.org/Predicate",
  "Literal value"
)
```

Data can be loaded from file by providing a file name and a format ("RDF/XML", "TURTLE", "N-TRIPLES", and "N3"):

```
store = load.rdf("file.n3", format="N3")
```

3.1.1 Example

As an example, we will here create RDF for a small data set with information on five single nucleotide polymorphisms (SNPs) in the BRCA1 gene, extracted with biomaRt [12]:

```
library(biomaRt)
mart = biomaNoy2009Rt::useMart(biomart="snp", dataset="hsapiens_snp")
brca1 = c("rs16940","rs16941", "rs16942", "rs799916", "rs799917")
attribs = c(
  "refsnp_id", "chr_name", "chrom_start",
  "validated", "pmid_20137", "allele"
)
data = biomaRt::getBM(
  attributes=attribs, filters=c("snp_filter"), values=brca1, mart=mart
)
```

Following the guidelines for generating RDF outlined by Open PHACTS [13, 4], we identify concepts and matching ontology terms, using BioPortal [14]: SNP, gene, chromosome, article, and allele. Additionally, there are the following properties: a SNP identifier, a PubMed identifier, a chromosome number, a chromosomal sequence position, and a list of validations (e.g. HapMap, 1000Genomes). The latter will not be further semantically specified, but just be converted into a RDF literal as it is provided by the BioMart.

PeerJ PrePrints | http://dx.doi.org/10.7287/peerj.preprints.185v2 | CC-BY 3.0 Open Access | received: 2 Feb 2014, published: 2 Feb 2014

Table 1: A list of ontology terms for the concepts in the BRCA1 SNP example. Ontology acronyms and the prefixes used are explained in the text.

concept	ontology	term	URI
SNP	SIO	snp	sio:SIO_010027
chromosome	GO	chromosome	obo:GO_0005694
article	BIBO	article	bibo:Article
allele	SO	allele	obo:SO_0001023
SNP identifier	TMO	snp identifier	tmo:TMO_0161
PubMed identifier	CHEMINF	PubMed identifier	sio:CHEMINF_000302
chromosome number	TMO	chromosome number	tmo:TMO_0157
chromosomal sequence position	TMO	chromosomal sequence position	tmo:TMO_0122

Taking advantage of existing ontologies. The choice is somewhat arbitrary and no ontological analysis has been done, e.g. on specified domain and ranges of predicates and implied properties of classes. The following ontologies are used: SIO is the Semanticscience Integrated Ontology; TMO is the Translational Medicine Ontology [15]; GO is the Gene Ontology; BIBO is the Bibliographic Ontology [16]; and, CiTO is the Citation Typing Ontology [17].

Prefixes used in the term Uniform Resource Identifiers (URIs) include *sio* for `http://semanticscience.org/resource/`, *tmo* for `http://www.w3.org/2001/sw/hcls/ns/transmed/`, *obo* for `http://purl.obolibrary.org/obo/`, *bibo* for `http://purl.org/ontology/bibo/`, and *cito* for `http://purl.org/spar/cito/`. We can add these prefixes to the store. The following code examples shows the full syntax in the first addition, but leaves out the parameters names in the following additions:

```
snpStore = new.rdf(ontology=FALSE)
add.prefix(snpStore,
  prefix="sio", namespace="http://semanticscience.org/resource/"
)
add.prefix(snpStore, "tmo", "http://www.w3.org/2001/sw/hcls/ns/transmed/")
add.prefix(snpStore, "obo", "http://purl.obolibrary.org/obo/")
add.prefix(snpStore, "bibo", "http://purl.org/ontology/bibo/")
add.prefix(snpStore, "cito", "http://purl.org/spar/cito/")
add.prefix(snpStore, "snp", "http://example.org/snp/")
add.prefix(snpStore, "art", "http://example.org/article/")
add.prefix(snpStore, "loc", "http://example.org/location/")
add.prefix(snpStore, "all", "http://example.org/allele/")
add.prefix(snpStore, "ex", "http://example.org/onto/")
add.prefix(snpStore, "pubmedid", "http://example.org/pubmed/")
add.prefix(snpStore, "snpid", "http://example.org/snpid/")
```

Furthermore, we can define a number of classes and predicate, for reduced code:

```
snpClass = "http://semanticscience.org/resource/SIO_010027"
chromosomeClass = "http://purl.obolibrary.org/obo/GO_0005694"
pubmedIdClass = "http://semanticscience.org/resource/CHEMINF_000302"
snpIdClass = "http://www.w3.org/2001/sw/hcls/ns/transmed/TMO_0161"
articleClass = "http://purl.org/ontology/bibo/Article"
```


PeerJ PrePrints | http://dx.doi.org/10.7287/peerj.preprints.185v2 | CC-BY 3.0 Open Access | received: 2 Feb 2014, published: 2 Feb 2014

```
alleleClass = "http://purl.obolibrary.org/obo/SO_0001023"

rdfTypePred = "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"
isDescribedBy = "http://purl.org/spar/cito/isDescribedBy"
onChromosome = "http://example.org/onto/onChromosome"
hasAttribute = "http://semanticscience.org/resource/CHEMINF_000200"
hasStart = "http://example.org/onto/hasStart"
hasValue = "http://semanticscience.org/resource/SIO_000300"
hasValidation = "http://example.org/onto/hasValidation"
hasAllele = "http://example.org/onto/hasAllele"
hasLocation = "http://example.org/onto/hasLocation"
```

Using these ontology terms we can define the translations of the BioMart-extracted data into RDF:

```
createEntry <- function(row) {
  snpID = row[1]
  chr = row[2]
  chrStart = row[3]
  validation = row[4]
  pubmedID = row[5]
  allele = row[6]

  snpSubject = paste("http://example.org/snp/", snpID, sep="")
  pubmedObject = paste("http://example.org/article/a", pubmedID, sep="")
  alleleObject = paste("http://example.org/allele/", snpID, sep="")
  chrObject = paste("http://example.org/location/", snpID, sep="")

  snpIDObject = paste("http://example.org/snpid/", snpID, sep="")
  pubmedIDObject = paste("http://example.org/pubmed/a", pubmedID, sep="")

  add.triple(snpStore, snpSubject, rdfTypePred, snpClass)
  add.triple(snpStore, snpSubject, hasAttribute, snpIDObject)
  add.triple(snpStore, snpIDObject, rdfTypePred, snpIdClass)
  add.data.triple(snpStore, snpIDObject, hasValue, snpID)
  # validation information
  sapply(unlist(strsplit(validation, split=",")),
    function(validationItem) {
      if (!is.null(validationItem)) {
        add.data.triple(snpStore,
          snpSubject, hasValidation, validationItem
        )
      }
    }
  )
  # allele information
  add.triple(snpStore, snpSubject, hasAllele, alleleObject)
  add.triple(snpStore, alleleObject, rdfTypePred, alleleClass)
  add.data.triple(snpStore, alleleObject, hasValue, allele)
  # chromosome location information
  add.triple(snpStore, snpSubject, hasLocation, chrObject)
  add.triple(snpStore, chrObject, rdfTypePred, chromosomeClass)
  add.data.triple(snpStore, chrObject, onChromosome, chr)
  add.data.triple(snpStore, chrObject, hasStart, chrStart)
```


PeerJ PrePrints | http://dx.doi.org/10.7287/peerj.preprints.185v2 | CC-BY 3.0 Open Access | received: 2 Feb 2014, published: 2 Feb 2014

```
  # pubmed information
  add.triple(snpStore, snpSubject, isDescribedBy, pubmedObject)
  add.triple(snpStore, pubmedObject, rdfTypePred, articleClass)
  add.triple(snpStore, pubmedObject, hasAttribute, pubmedIDObject)
  add.triple(snpStore, pubmedIDObject, rdfTypePred, pubmedIdClass)
  add.data.triple(snpStore, pubmedIDObject, hasValue, pubmedID)
}
```

We can then just iterate over the actual data with and safe it to a file:

```
apply(data, MARGIN=1, FUN=createEntry)
save.rdf(snpStore, filename="test.n3", format="N3")
```

Or output it as a Notation3 string with:

```
cat(asString.rdf(snpStore))
```

3.2 SPARQL

Local (in-memory) triple stores can be queried with the *sparql.rdf* method by providing it with the store to query and a string with the SPARQL query. For example, to get all resource types, we can use this:

```
sparql.rdf(store,
  paste(
    "SELECT DISTINCT ?type WHERE {",
    " [] a ?type ",
    "}"
  )
)
```

Remote SPARQL end point can be queried in a similar fashion: we replace the triple store with the URL pointing to the SPARQL end point. For example, if we want to extract all properties of the CHEMBL615603 assay from ChEMBL [18, 5] we can use the following remote query:

```
results = sparql.remote(
  "http://rdf.farmbio.uu.se/chembl/sparql",
  paste(
    "SELECT DISTINCT ?predicate ?object WHERE {",
    " ?assay <http://www.w3.org/2000/01/rdf-schema#label> \"CHEMBL615603\" ;",
    "        ?predicate ?object .",
    "}"
  )
)
```

Jena parses the SPARQL query too, which requires the query to be valid against both Jena and the SPARQL end point. Because the SPARQL specification has tool specif extensions and that not all tools support the full of SPARQL 1.1, it can sometimes be tricky to find the mutually support SPARQL command subset. The rrdf package also allows to bypass

PeerJ PrePrints | http://dx.doi.org/10.7287/peerj.preprints.185v2 | CC-BY 3.0 Open Access | received: 2 Feb 2014, published: 2 Feb 2014

Jena and query the end point directly by using *sparql.remote(..., jena=FALSE)*, removing the problem and allowing you to use extensions of the end point.

The results object is a matrix with the variable names from the SPARQL query as column names. This allows us to get a predicates with:

```
results[,"predicate"]
```

3.2.1 Example

Remote SPARQLing can be used to get more information about biological entities of interest. The following example retrieves information from the Uniprot database using the experimental SPARQL end point. In particular, it retrieves annotations for proteins encoded by the BRCA1 gene:

```
findAnnotations = paste(
  "prefix up: <http://purl.uniprot.org/core/>",
  "prefix ens: <http://purl.uniprot.org/ensembl/>",
  "prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>",
  "SELECT DISTINCT ?protein ?mnemonic ?comment WHERE {",
  "  ?transcript up:transcribedFrom ens:ENSG00000012048 .",
  "  ?protein a up:Protein ;",
  "    rdfs:seeAlso ?transcript ;",
  "    up:mnemonic ?mnemonic ;",
  "    up:annotation ?annot .",
  "  ?annot rdfs:comment ?comment .",
  "}"
)
annotations = sparql.remote(
  endpoint="http://beta.sparql.uniprot.org/",
  sparql=findAnnotations, jena=FALSE
)
```

3.2.2 Example

A second example combines regular REST or -like services that return RDF with a local SPARQL query. For example, this is the approach used in Bioclipse to interact with many of the OpenTox services [5], but can be used for the Open PHACTS Linked Data API too [4]. For example, we can get data from an OpenTox data set using RCurl [19, 20], in this example data set 112 with 320 compounds, which originates from the ToxCast project [21]:

```
library(RCurl)
rdfContent = getURL(
  paste(
    "http://apps.ideaconsult.net:8080/ambit2/dataset/112/compounds",
    "media=text/n3",
    sep="?"
  ),
  write=basicTextGatherer()
)
store = fromString.rdf(rdfContent, format="N3")
compounds = sparql.rdf(store,
```


PeerJ PrePrints | http://dx.doi.org/10.7287/peerj.preprints.185v2 | CC-BY 3.0 Open Access | received: 2 Feb 2014, published: 2 Feb 2014

```
  paste(
    "PREFIX ot: <http://www.opentox.org/api/1.1#>",
    "SELECT DISTINCT ?compound WHERE {",
    " ?compound a ot:Compound",
    "}"
  )
)
```

In fact, we can continue getting further detail about these compound by repeating the same approach on the compound URIs. Let's assume we are further interested in the SMILES string. This feature has the URI `http://apps.ideaconsult.net:8080/ambit2/feature/21753`. We add this as the list of features to retrieve:

```
feature = "http://apps.ideaconsult.net:8080/ambit2/feature/21753"
getOptions = paste(
  paste("feature_uris[]", curlEscape(feature), sep="="),
  paste("media", curlEscape("application/rdf+xml"),sep="="),
  sep="&"
)
```

We can then use the R apply() method to retrieve the SMILES for each structure with:

```
result = apply(compounds, MARGIN=1, function(x) {
  compound = x["compound"];
  compoundURI = paste(compound, getOptions, sep="?")
  print(paste("Downloading",compoundURI))
  cmpdContent = getURL(
    compoundURI, write=basicTextGatherer()
  )
  fromString.rdf(cmpdContent, format="RDF/XML", appendTo=store)
})
```

Finally, we use SPARQL again to list all the SMILES strings for the compounds:

```
query =  paste(
  "PREFIX ot: <http://www.opentox.org/api/1.1#> ",
  "SELECT DISTINCT ?compound ?value WHERE {",
  " ?s ?o [ ot:feature <", feature, "> ;",
  "   ot:value ?value ",
  " ] ;",
  "   ot:compound ?compound . ",
  "}",
  sep=""
)
compoundSMILESes = sparql.rdf(store, query)
```

This results in a matrix with compound URIs in one column and SMILES strings in another, ready for use by other R packages.

PeerJ PrePrints | http://dx.doi.org/10.7287/peerj.preprints.185v2 | CC-BY 3.0 Open Access | received: 2 Feb 2014, published: 2 Feb 2014

4 Conclusions

The rrdflibs and rrdf packages bring basic semantic web technologies to R statistical environment. While only a subset of Apache Jena is currently exposed, it provides key methods to deal with RDF data and resources. The examples shed some light on how it can find a place on more advanced analyses workflows.

5 Acknowledgements

The people who have provided feedback on the rrdf package are kindly thanked for their efforts in getting into contact with me. The research leading to these results has received funding from the European Communitys Seventh Framework Programme (FP7/20072013) under Grant Agreement n [267042], from Cosmetics Europe, and from the Innovative Medicines Initiative Joint Undertaking under grant agreement no. 115191, resources of which are composed of financial contribution from the European Union's Seventh Framework Programme (FP7/2007-2013) and EFPIA companies in-kind contribution.

References

[1] F. Belleau et al. "Bio2RDF: Towards a mashup to build bioinformatics knowledge systems". In: *Journal of Biomedical Informatics* 41.5 (Oct. 21, 2008), pp. 706–716. ISSN: 15320464. DOI: 10.1016/j.jbi.2008.03.004. URL: http://dx.doi.org/10.1016/j.jbi.2008.03.004.

[2] B. Chen et al. "Chem2Bio2RDF: a semantic framework for linking and data mining chemogenomic and systems chemical biology data". In: *BMC Bioinformatics* 11.1 (May 17, 2010), pp. 255+. ISSN: 1471-2105. DOI: 10.1186/1471-2105-11-255. URL: http://dx.doi.org/10.1186/1471-2105-11-255.

[3] M. Samwald et al. "Linked open drug data for pharmaceutical research and development". In: *Journal of Cheminformatics* 3.1 (2011), pp. 19+. ISSN: 1758-2946. DOI: 10.1186/1758-2946-3-19. URL: http://dx.doi.org/10.1186/1758-2946-3-19.

[4] A. J. Williams et al. "Open PHACTS: semantic interoperability for drug discovery". In: *Drug Discovery Today* 17.21-22 (Nov. 7, 2012), pp. 1188–1198. ISSN: 13596446. DOI: 10.1016/j.drudis.2012.05.016. URL: http://dx.doi.org/10.1016/j.drudis.2012.05.016.

[5] E. Willighagen et al. "The ChEMBL database as linked open data". In: *Journal of Cheminformatics* 5.1 (2013), pp. 23+. ISSN: 1758-2946. DOI: 10.1186/1758-2946-5-23. URL: http://dx.doi.org/10.1186/1758-2946-5-23.

[6] *EBI RDF Platform*. 2013. URL: http://www.ebi.ac.uk/rdf/.

[7] A. Seaborne and S. Harris. *SPARQL 1.1 Query*. W3C Working Draft. W3C, Oct. 2009. URL: http://www.w3.org/TR/2009/WD-sparql11-query-20091022/.

PeerJ PrePrints | http://dx.doi.org/10.7287/peerj.preprints.185v2 | CC-BY 3.0 Open Access | received: 2 Feb 2014, published: 2 Feb 2014

[8] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: http://www.R-project.org/.

[9] *Apache Jena 2.11*. 2013. URL: http://jena.apache.org/.

[10] S. Urbanek. *rJava - Low-level R to Java interface*. Dec. 3, 2013. URL: http://cran.r-project.org/web/packages/rJava/.

[11] W. R. Van Hage. *SPARQL: SPARQL client*. http://cran.r-project.org/web/packages/SPARQL/. Oct. 25, 2013.

[12] S. Durinck et al. "BioMart and Bioconductor: a powerful link between biological databases and microarray data analysis". In: *Bioinformatics* 21.16 (Aug. 15, 2005), pp. 3439–3440. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bti525. URL: http://dx.doi.org/10.1093/bioinformatics/bti525.

[13] C. Haupt et al. *Guidelines for exposing data as RDF in Open PHACTS*. Tech. rep. Oct. 7, 2013. URL: http://www.openphacts.org/specs/2013/WD-rdfguide-20131007/.

[14] N. F. Noy et al. "BioPortal: ontologies and integrated data resources at the click of a mouse". In: *Nucleic Acids Research* 37.suppl 2 (July 1, 2009), W170–W173. ISSN: 1362-4962. DOI: 10.1093/nar/gkp440. URL: http://dx.doi.org/10.1093/nar/gkp440.

[15] J. Luciano et al. "The Translational Medicine Ontology and Knowledge Base: driving personalized medicine by bridging the gap between bench and bedside". In: *Journal of Biomedical Semantics* 2.Suppl 2 (2011), S1+. ISSN: 2041-1480. DOI: 10.1186/2041-1480-2-s2-s1. URL: http://dx.doi.org/10.1186/2041-1480-2-s2-s1.

[16] B. D'Arcus and F. Giasson. "Bibliographic Ontology Specification. Specification Document, 4 November 2009". In: 10 (2009), p. 2011.

[17] D. Shotton. "CiTO, the Citation Typing Ontology". In: *Journal of Biomedical Semantics* 1.Suppl 1 (2010), S6+. ISSN: 2041-1480. DOI: 10.1186/2041-1480-1-s1-s6. URL: http://dx.doi.org/10.1186/2041-1480-1-s1-s6.

[18] A. Gaulton et al. "ChEMBL: a large-scale bioactivity database for drug discovery". In: *Nucleic Acids Research* 40.D1 (Jan. 1, 2012), pp. D1100–D1107. ISSN: 1362-4962. DOI: 10.1093/nar/gkr777. URL: http://dx.doi.org/10.1093/nar/gkr777.

[19] B. Hardy et al. "Collaborative development of predictive toxicology applications". In: *Journal of Cheminformatics* 2.1 (Aug. 31, 2010), pp. 7+. ISSN: 1758-2946. DOI: 10.1186/1758-2946-2-7. URL: http://dx.doi.org/10.1186/1758-2946-2-7.

[20] D. T. Lang. *RCurl: General network (HTTP/FTP/...) client interface for R*. R package version 1.95-4.1. 2013. URL: http://CRAN.R-project.org/package=RCurl.

[21] D. J. Dix et al. "The ToxCast Program for Prioritizing Toxicity Testing of Environmental Chemicals". In: *Toxicological Sciences* 95.1 (Jan. 1, 2007), pp. 5–12. ISSN: 1096-6080. DOI: 10.1093/toxsci/kfl103. URL: http://dx.doi.org/10.1093/toxsci/kfl103.

PeerJ PrePrints | http://dx.doi.org/10.7287/peerj.preprints.185v2 | CC-BY 3.0 Open Access | received: 2 Feb 2014, published: 2 Feb 2014