Programmatic retrieval of information from DAS servers using the **DASiR** package

Oscar Flores Guri and Anna Mantsoki
Institute for Research in Biomedicine & Barcelona Supercomputing Center
Joint Program on Computational Biology

April 12, 2014

1 Introduction

Distributed Annotation System (DAS) is a protocol for information exchange between a server and a client. Is widely used in bioinformatics and the most important repositories include a DAS server parallel to their main front-end. Few examples are "UCSC", "Ensembl" or "UniProt".

The DASiR package provides a convenient R-DAS interface to programmatically access DAS servers available from your network. It supports the main features of DAS 1.6 protocol providing a convenient interface to R users to a huge amount of biological information. DAS uses XML and HTTP protocols, so the server deployment and client requirements are significantly less than MySQL or BioMart based alternatives. You can find an browsable list with more than 1500 on-line DAS servers in the url `http://www.dasregistry.org`.

Despite the DAS protocol supports querying different kinds of data, DASiR has been designed with ranges-features in mind. Querying genomic sequences and protein structures is also supported, but you probably will find better ways to access such data in R if you requiere an intensive use of them (Biostrings genomes or Bio3dD package for PDB structure, for example).

Here you have a brief summary of the main functions of DASiR:

- Set/get the DAS server: `setDasServer`, `getDasServer`
- Retrieve the data sources: `getDasSource`, `getDasDsn`
- Retrieve the entry points and types: `getDasEntries`, `getDatTypes`
- Retrieve information about the features: `getDasFeature`
- Nucleotide/amino acid sequence retrieval: `getDasSequence`
- Protein 3D structure retrieval: `getDasStructure`

For detailed information about these functions and how to use them refer to the DASiR manual or give a look to the following sections for an overview.

2 DAS metadata information handling

As an important part of the dialog between the client and the server consists on knowing which information has every server, DASiR provides the following functions to query metadata from a DAS server.

The first step before start quering and retrieving information is, of course, set the DAS server we will use during this session. Notice that DASiR only supports one active DAS session per R instance.

To set the server we will use the function `setDasServer`:

```
> setDasServer(server="http://genome.ucsc.edu/cgi-bin/das")
> getDasServer()
```

```
[1] "http://genome.ucsc.edu/cgi-bin/das"
```

The function `getDasSource()` will return the id's, the titles and the capabilities of the data sources available in the server in the form of a data frame. This function is important since it is necessary to use the exact name ("id" or "title" depending on the server) as reference name in the other functions of the package.

```
> #sources = getDasSource() #This will fail for UCSC (but no for ENSEMBL)
> sources = getDasDsn() #This will fail for ENSEMBL (but no for UCSC)
> head(sources)
```

```
[1] "hg38"    "hg19"    "hg18"    "hg17"    "hg16"    "panTro4"
```

We should also take into account that the values in capabilities of each data source returned by this function. If we query an unknown name or we query for a capability that is not implemented in the server (for example we ask for a atomic structure to a sequence server) DASiR will return a NULL value, but due to the nature of HTTP based queries, we cannot detect the origin of this error without overloading the server with cross-calls.

Despite "sources" is the common way to recover server features, some servers (as UCSC) are still using the deprecated "dsn" option to get the ids of the different datasets. The rule of a thumb is that if the first don't work, try the second.

Once we have the name of the database we want to query, we need to know where we can look. This is called "entry point" and could be from a protein ID to a chromosome number (depending on the type of server, of course). Notice that output will be a GRanges by default (if the server supplies start, stop and id values). If this is not possible or textttas.GRanges=FALSE, the output will be a data.frame.

```
> source = "sacCer3"
> entries = getDasEntries(source, as.GRanges=TRUE)
> head(entries)
```

```
GRanges with 6 ranges and 2 metadata columns:
      seqnames       ranges strand | orientation subparts
         <Rle>    <IRanges>  <Rle> |    <factor> <factor>
  [1]       IV [1, 1531933]      * |           +       no
  [2]       XV [1, 1091291]      * |           +       no
  [3]      VII [1, 1090940]      * |           +       no
  [4]      XII [1, 1078177]      * |           +       no
  [5]      XVI [1,  948066]      * |           +       no
  [6]     XIII [1,  924431]      * |           +       no
  ---
  seqlengths:
      I   II  III   IV   IX    M    V ...    X   XI  XII XIII  XIV   XV  XVI
     NA   NA   NA   NA   NA   NA   NA ...   NA   NA   NA   NA   NA   NA   NA
```

Finally, the different attributes we can ask for, called "types" could be obtained with the function `getDasTypes`.

```
> types = getDasTypes(source)
> head(types)
```

```
[1] "blastHg18KG"      "est"              "intronEst"        "mrna"
[5] "ensGene"          "esRegGeneToMotif"
```

Think about Sources as the name of the database, Entries are the tables of this database and finally Types are the columns on this table.

In general a `character` vector is returned for most of the functions, except for `getDasSource` and `getDasEntries` which return a data.frame with the name of the entry the ranges of the elements and possibly other attributes depending on the server.

3 Querying features

The `getDasFeature` function queries the DAS server and returns the available information for the type(s) at the given range(s).

```
> ranges=entries[c(1,2)]
> types=c("sgdGene","mrna")
> features = getDasFeature(source, ranges, types)
> head(features)
```

```
  segment.range                     id    label type method  start    end score
1             1 AY169693.chrIV.84188.0 AY169693 mrna   BLAT  84189  85450   990
2             1 AY169693.chrIV.84188.1 AY169693 mrna   BLAT 520477 520627   990
3             1 AY169693.chrIV.84188.2 AY169693 mrna   BLAT 520628 520633   990
4             1 AY169693.chrIV.84188.3 AY169693 mrna   BLAT 520634 520643   990
5             1 AY169693.chrIV.84188.4 AY169693 mrna   BLAT 540324 540328   990
```

```
6             1 AY169693.chrIV.84188.5 AY169693 mrna   BLAT 593167 593177   990
  orientation phase                        group
1           +     - Link to UCSC BrowserAY169693
2           +     - Link to UCSC BrowserAY169693
3           +     - Link to UCSC BrowserAY169693
4           +     - Link to UCSC BrowserAY169693
5           +     - Link to UCSC BrowserAY169693
6           +     - Link to UCSC BrowserAY169693
```

A `data.frame` is returned with the contents of the specific annotation for the given ranges and types in the server (again, note that servers could provide different/additional information for types with the same name).

4 Querying nucleotide or amino acid sequence

The `getDasSequence` function queries the DAS server and retrieves the nucleotide or amino acid sequence for the given ranges.

```
> #Now we will retrieve sequences from VEGA server
> setDasServer("http://vega.sanger.ac.uk/das")
> source = "Homo_sapiens.VEGA51.reference"
> ranges = GRanges(c("1","2"), IRanges(start=10e6, width=1000))
> #Returning character vector, we only ask 50 first bases in the range
> sequences = getDasSequence(source, resize(ranges, fix="start", 50))
> print(sequences)
```

```
[1] "aaccccgtctctacaataaattaaaatattagctgggcatggtggtgtgt"
[2] "gtattagtttgtttccacactactatgaagatactacctgagactgggta"
```

An `character` vector is returned (the default class) containing the nucleotide or amino acid sequences that match the content of the annotation for the given ranges in the server. Automatic conversion to Biostring classes is supported with the `class` attribute:

```
> #Now we specify we want a AAStringSet (Biostring class for AminoAcids strings)
> #and query for the whole sequence length
> sequences = getDasSequence(source, ranges, class="AAStringSet")
> print(sequences)
```

```
  A AAStringSet instance of length 2
    width seq
[1]  1000 aaccccgtctctacaataaattaaaatattagct...aattttccagtaaaagtcaaggcaaacccaaga
[2]  1000 gtattagtttgtttccacactactatgaagatac...tagactgagaggagagaaattggagacaacaaa
```

5 Query a protein 3D structure

The `getDasStructure` function queries the DAS server and retrieves the 3D structure, including metadata and coordinates for the given query (ID of the reference structure) using the data source id (or title).

```
> #On 2013-03-05 there are 2 structure servers in dasregistry.org...
> setDasServer(server="http://das.sanger.ac.uk/das")
> #Get the sources with "structure" capability...
> sources = getDasSource()
> sources[grep("structure", sources$capabilities),]
```

```
           id     title   capabilities
300 structure structure das1:structure
```

```
> source="structure" #...which name is also "structure"
> query="1HCK" #PDB code
> structure=getDasStructure(source,query)
> head(structure)
```

```
  atomID atomName       x       y      z  type groupID name
1      1       N 102.329 111.862 92.452 amino       1  MET
2      2      CA 103.332 112.165 93.516 amino       1  MET
3      3       C 103.877 113.584 93.255 amino       1  MET
4      4       O 103.802 114.075 92.129 amino       1  MET
5      5      CB 104.437 111.099 93.495 amino       1  MET
6      6      CG 105.176 110.881 94.812 amino       1  MET
```

An `data.frame` is returned with the information of every atom of the reference structure of the given query. Notice that a single residue (identified by groupID and name) has few atoms inside. Depending on your purposes, you can find this way to represent the structural information unconvenient (for example, if you are used to work with PDB format). Structure query in DAS seems a secondary feature of the protocol (as only 2 servers out of >1500 are supporting it) but anyway we wanted to support the maximum amount of features that DAS protocol implement. We realize that are better options to work with PDB files, but maybe this one can help you for quick structural analysis.

6 Plotting of obtained features

The `plotFeatures` function creates a basic plot with the features retrieved by `getDasFeature` or adds them to an existing plot.

```
> #Let's retrieve some genes from UCSC Genome Browser DAS Server now
> setDasServer(server="http://genome.ucsc.edu/cgi-bin/das")
> #Official yeast genes and other annotated features in the range I:22k-30k
```

```
> source = "sacCer3" #Saccharomices Cerevisiae
> range = GRanges(c("I"), IRanges(start=22000, end=30000))
> type  = c("sgdGene", "sgdOther") #This is also the name of the UCSC tracks
> features = getDasFeature(source, range, type)
> #Only the main columns
> head(features[,c("id", "label", "type", "start", "end")])
```

```
                      id      label     type start   end
1 YAL063C-A.chrI.22394.0  YAL063C-A  sgdGene 22395 22685
2   YAL063C.chrI.23999.0    YAL063C  sgdGene 24000 27968
3  YALWdelta1.chrI.22230 YALWdelta1 sgdOther 22231 22552
4        FLO9.chrI.24000       FLO9 sgdOther 24001 27968
```

```
> plotFeatures(features, box.height=10, box.sep=15, pos.label="top", xlim=c(22000,30000))
```

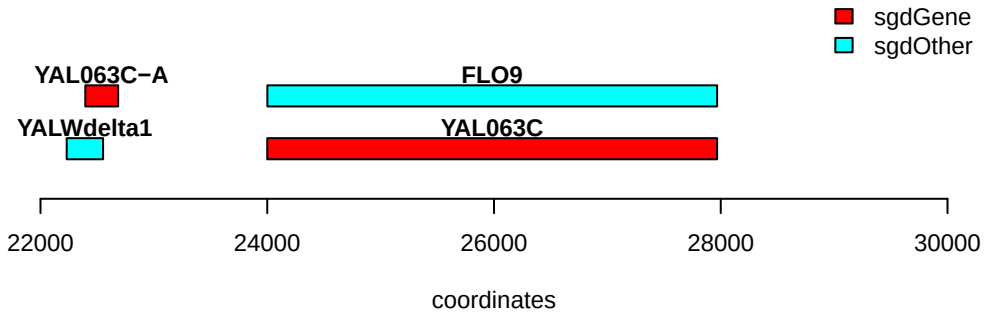

Figure 1: Basic feature plot generated with `getDasFeature` and `plotFeatures`

The `plotFeatures` function is a simple way of drawing text boxes in a new or an existing plot with the features retrieved. Notice that when overplotting to a already open graphical device, the x-coordinates must match.

And this is all. You can find more detailed information and examples of each function in the R manpages for DASiR. We hope this package helps you in your data mining.