HOKKAIDO UNIVERSITY

Title	Gfdnavi, Web-Based Data and Knowledge Server Software for Geophysical Fluid Sciences, Part II: RESTful Web Services and Object-Oriented Programming Interface
Author(s)	Nishizawa, Seiya; Horinouchi, Takeshi; Watanabe, Chiemi; Isamoto, Yuka; Tomobayashi, Akinori; Otsuka, Shigenori; GFD Dennou Club
Citation	Lecture notes in computer science, 6193, 105-116 https://doi.org/10.1007/978-3-642-14589-6_11 Database Systems for Advanced Applications 15th International Conference, DASFAA 2010, International Workshops: GDM, BenchmarX, MCIS, SNSMW, DIEW, UDM, Tsukuba, Japan, April 1-4, 2010, Revised Selected Papers, ISBN:978-3-642-14588-9
Issue Date	2010-08
Doc URL	http://hdl.handle.net/2115/45818
Rights	The final publication is available at www.springerlink.com
Type	article (author version)
File Information	prepri_gfdnavi2010part2_LNCS.pdf

Instructions for use

Hokkaido University Collection of Scholarly and Academic Papers : HUSCAP

Gfdnavi, Web-based Data and Knowledge Server Software for Geophysical Fluid Sciences, Part II: RESTful Web Services and Object-Oriented Programming Interface

Seiya Nishizawa[1], Takeshi Horinouchi[2], Chiemi Watanabe[3], Yuka Isamoto[3], Akinori Tomobayashi[4], Shigenori Otsuka[5], and GFD Dennou Club[6]

[1] Department of Earth and Planetary Sciences, Kobe University, 1-1 Rokohdai-cho Nada-ku Kobe, Hyogo 657-8501, Japan
[2] Faculty of Environmental Earth Science, Hokkaido University
[3] Department of Information Sciences, Ochanomizu University
[4] Shoganji
[5] Department of Geophysics, Kyoto University
[6] www.gfd-dennou.org

Abstract. In recent years, increasing amounts of scientific data on geophysical and environmental fluids, e.g., in the atmosphere and oceans, are being available. Further, there is increasing demand for web-based data services. Several browser-based data servers, on which geophysical-fluid data can be analyzed and visualized, have been developed. However, they are suitable only for initial "quick-looks" and not for subsequent research processes. As a solution, we developed data server software named Gfdnavi. One of its important features is that it provides extensive support for programming (scripting). With Gfdnavi, users can easily switch between operations using a web browser and operations using scripts or command lines. This paper describes its network features: web services, which is an important part of Gfdnavi's programmability, and the functionality to search across multiple Gfdnavi servers. To develop the web services, we adopted the REST architecture. We also developed a client library to ensure access to web services in the programming language Ruby. Using this library, data can be analyzed and visualized on either the server side or client side. It also enables data handling on multiple servers. Search across multiple web servers is made possible by a simple peer-to-peer network with a central server, with the peer-to-peer communication based on web services.

1 Introduction

In recent years increasing amounts of scientific data on geophysical and environmental fluids, e.g., in the atmosphere and oceans, are being available. Many data centers and research organizations or groups are now providing data through the Internet, and some are also providing on-line visualization capabilities.

As stated in the companion paper [1] (hereafter referred to as "Part I"), the following problems are encountered by existing web-based data servers during browser-based access employed in geophysical fluid sciences:

1. The visualization capability of existing web-based data servers is limited, because of which only initial "quick-looks" are possible; the resultant diagrams would not be of sufficiently high quality suitable for publication.
2. The features of the servers are not available once the data files are downloaded. Therefore, the data need to be analyzed and visualized independently. In such a case, even the opening of files could be difficult, because a number of binary data formats are used in these sciences.
3. Most of the browser-based advanced data servers support only georeferencing data, despite the fact that non-georeferencing data are also important and frequently required for conducting researches.
4. The search capability of these servers is often limited. Furthermore, it is difficult to search for data across networks, except for documents that are available via Internet search engines.
5. Interdisciplinary and/or collaborative studies would require communication among researchers for the purpose of exchanging know-hows, results, and so on; however, to the best of the authors' knowledge, none of the existing data servers for geophysical fluid sciences support such communication.

To solve all these problems, we have developed web-based data and knowledge server software for application in geophysical fluid sciences, termed Gfdnavi [2]. The basic features of Gfdnavi are described in Part I. Part II of this paper deals mainly with the solution to problems 1 and 4 by the use of the developed Gfdnavi. It is, however, also useful for solving problem 2.

In order to ensure the usefulness of data servers in all stages of scientific studies, it is necessary to provide programmability to clients. A browser-based graphical user interface (GUI) would be useful in initial trial-and-error stages; however, it would not necessarily be productive in the later stages, in which repetition is frequently required; instead, programming would be effective for such repetition. Therefore, the support of a smooth transition between GUI operations and programming by a data server would be a useful feature.

In order to improve the practical applicability of Gfdnavi, we have included in it multiple ways of programmability; as a result, it can be used to

1. support web services
2. allow the user to download a subset of data and a script for reproducing the action that he or she performed on the server with the GUI
3. allow a registered user to upload scripts to carry out his/her original data analysis and visualization on the server

The web services (item 1 in the above list) are described in the present paper, whereas the other two items have been described in Part I.

We have not only included support for web services in Gfdnavi but also provided a client library for accessing the web services in the object-oriented

programming (hereafter referred to as "OOP") language Ruby [3]. As will be shown subsequently, because of the library, the programmability associated with Gfdnavi is unified irrespective of the data location (e.g., on the run-time memory, in locally accessible external files, or over the Internet) and access method (e.g., through Gfdnavi, other network data services, or local input-output calls). This unification is the basis for achieving a smooth transition between GUI operations and programming.

Researchers in the field of geophysical fluid sciences frequently combine data from multiple sources for conducting research. It is, therefore, important to support the combining by realizing search and analysis across multiple data servers. However, existing browser-based data servers have limited capability in this respect. In particular, searching across networks (hereafter referred to as "cross search") is rather important, since researchers must first know what kinds of data are available for the task at hand and also the location of the data.

It would be ideal to conduct a search over a variety of heterogeneous data servers. Our scope, however, is limited to a search among Gfdnavi servers. Such a limited search would still be useful if Gfdnavi is used extensively. A typical use case that we aim to cover is as follows: Suppose that a researcher conducts a study using data obtained by his/her own observations or numerical simulations or those of his/her research group. In many cases, the researcher would additionally use external data such as satellite observations, which may be stored locally or available only on servers of data centers. In such a case, a search should be conducted across data on both local and remote servers.

In this paper, we present the design and implementation of two network features of Gfdnavi: web services and cross search. The rest of this paper is organized as follows. In Sect. 2, we present the design of Gfdnavi web services. Then, we describe the implementations of the web services in Sect. 3, a client library in Sect. 4, and functionality of use of multiple servers in Sect. 5. Finally, we present the conclusions and discussions in Sect. 6.

2 Design of Gfdnavi Web Services

2.1 Motivating Scenario

We show an example use case of Gfdnavi web services in research in the field of geophysical fluid sciences. Suppose that a researcher in the field having performed several numerical simulations of the future climate with different scenarios for future emission of carbon dioxide wants to compare these results. At first, he or she analyze and visualize result data of one simulation run out of the simulations with the GUI of Gfdnavi web applications, in order to figure out characteristics of spatial pattern of temperature. After that, he or she would want to apply the same analysis and visualization to result data of the other runs. Carrying out them with programming would be more efficient than that with GUI. A Ruby script reproducing the action performed with the GUI can be downloaded; e.g., Fig. 1. Methods in the script send an HTTP request to Uniform Resource Locator

(hereafter referred to as "URL") of Gfdnavi web services, which is as shown in Fig. 2. He or she modifies the script to perform the analysis and visualization with data of all the runs; e.g., Fig. 3.

Further, he or she would want to compare the results with those of simulations performed by other researchers. Using the cross search feature of Gfdnavi, he or she could find other simulation data than his/hers. Then the same analysis and visualization can be applied to these data, and comparison of those results can be performed.

```
1: require "numru/gfdnavi_data"
2: include NumRu
3: t = GfdnaviData.parse("http://example.com/data/T.exp01.nc/T")
4: t_mean = t.analysis("mean","longitude")
5: tone = t_mean.plot("tone")
6: png = tone.to_png
```

Fig. 1. Example of Ruby script reproducing the action with the GUI on Gfdnavi web applications.

```
line 3: /data/T.exp01.nc/T.yml
line 4: /data/T.exp01.nc/T/analysis(mean;longitude).yml
line 5: /data/T.exp01.nc/T/analysis(mean;longitude)/plot(tone).yml
line 6: /data/T.exp01.nc/T/analysis(mean;longitude)/plot(tone).png
```

Fig. 2. URL paths of HTTP requests yielded by methods in the Ruby script in Fig. 1. The number at left side of each line represents line number in the script.

2.2 Web Services and Client Library

We developed web services of Gfdnavi for ensuring programmability and a client library for programming irrespective of the data location and access method. We designed the web services such that they can be used from programs, particularly from the client library, and designed the client library such that it behaves in a manner similar to a library for data analysis and visualization at the local level.

RESTful Web Services Several technologies for developing web services are available, two of the popular ones being REST [4] and SOAP [5]. REST is based on resource-oriented architecture. It uses a uniform interface, i.e., an HTTP method. RESTful web services are stateless because of the use of HTTP methods.

```
 1: require "numru/gfdnavi_data"
 2: include NumRu
 3: NRUNS = 10  # number of runs
 4: pngs = Array.new
 5: for n in 0...NRUNS # loop for all the runs
 6:   crun = sprintf("%02d", n+1) #=> "01", "02", ...
 7:   t = GfdnaviData.parse("http://example.com/data/T.exp"+crun+".nc/T")
 8:   t_mean = t.analysis("mean","longitude")
 9:   tone = t_mean.plot("tone")
10:   pngs[n] = tone.to_png
11: end
```

Fig. 3. Example of Ruby script modified for repetition from the script in Fig. 1.

SOAP is a protocol for remote procedure call (RPC). Web services based on SOAP could be either stateful or stateless.

We decided to develop Gfdnavi web services with REST, the most important reason for this being that REST is based on resource-oriented architecture, which has similarities with OOP, and as a result, it would be easy to develop the Ruby client library having access to RESTful web services.

Statelessness is an important feature of RESTful web servers. A stateless system is advantageous over a stateful system in some ways. For example, scaling out a stateless system is easy. Scalability of Gfdnavi is important, because its operations such as analysis and visualization could require large computing resources such as CPU usage time and memory space. Further, testing of a stateless system is easier than that of a stateful system.

However, making Gfdnavi web services RESTful is somewhat difficult because of certain problems. One problem is how to define URL of dynamic resources, which are generated dynamically as result of operations such as analysis and visualization. In this study, we defined URLs such that they have correspondence with OOP. In OOP, dynamic objects generated from a static object are represented in the form `a_static_object.method1[.method2[....]]` with a method chain. In a similar manner, the URL path of the dynamic resources is defined as follows: `/a_static_resource/method1[/method2[/...]]`
The path of dynamic resources generated from multiple static resources is also defined on the basis of the correspondence of the URLs with OOP.

Another problem is dealing with temporary data in sequential programming. A program is usually a set of sequential operations. Results of operations are set to temporal variables and used later. In stateless web services, temporary resources are usually not used, though they are used to represent transactions in some systems. For simplicity, we decided not to use temporary resources in Gfdnavi. This could result in unnecessary duplicate executions. For example, an operation applied to a result of other operations could result in repeated executions of all those other operations. To prevent such duplication, we introduce delayed execution and cache mechanisms.

Ruby Client Library Many researchers in the field of geophysical fluid sciences use Ruby and a Ruby class library GPhys [6], which represents multi-dimensional numerical data and supports a variety of data formats, for data analysis and visualization at the local level. We developed a client library of Gfdnavi web services using Ruby in order to enable users to analyze and visualize data on Gfdnavi servers in a manner similar to programming with GPhys. Using the library, users can carry out data analysis and visualization on either the server side or the client side and select either side for execution, on the basis of efficiency or other factors.

2.3 Use of Multiple Servers

For researchers in the abovementioned field, who often work with several kinds of data, network capability is an important requirement, as explained in Sect. 1. Gfdnavi provides the features of cross search and analysis/visualization with multiple data provided by different Gfdnavi servers. The web services is used in Gfdnavi web applications, in order to access remote resources of other Gfdnavi servers.

Cross Search In the case of use of multiple servers for data analysis and visualization, the location of resources is specified by users. For cross search, however, users would want to search data on not only known servers but also unknown servers.

The following are some of the candidates for implementing cross search. One is a client-server model, which is a central server having all the information of all the data on servers forming a network for cross search; all the search requests are sent to the central server. Another is a pure peer-to-peer (P2P) network model, which contains all the information shared in the network. Yet another candidate is a hybrid P2P model, which is a server list managed by a central server; in this model, search requests are sent to each peer.

Generally, the pure P2P model has the advantage of scalability over the client-server model and does not require a high-performance central server. However, the development of the pure P2P model tends to be difficult. We decided that at least for the time being, the hybrid P2P model is suitable for Gfdnavi.

3 Implementation of Gfdnavi Web Services

In this section, we present the implementation of Gfdnavi web services.

There are two types of resources in Gfdnavi web services: static and dynamic. A static resource represents column data in database tables, such as data in files and knowledge data. These data are organized as a directory tree structure, and each data set is called a node. A dynamic resource represents results of operations such as search, analysis, or visualization.

3.1 URL Syntax

Each resource of Gfdnavi web services has at least one URL, which consists of prefix for Gfdnavi web service, resource path, extension, and parameters: `http://{host}:{port}/.../{resource_path}.{extension}?{params}`, where "`{var}`" is a variable name and is substituted to the value of the variable. "`Extension`" specifies the media type for representing the response for a requested resource (Table 1).

Table 1. Extensions for URL of Gfdnavi web service.

extension	media type
html	text/html (HTML)
xml	text/xml (XML)
yml	text/x-yaml (YAML [7])
nc	application/x-netcdf (NetCDF [8])
png	image/png (PNG)
knlge	text/x-yaml (YAML for Gfdnavi knowledge data)
gphys	application/octet stream (GPhys marshaled binary)

"`Params`" are parameters for representation, and they can be omitted along with the leading question mark. "`Resource_path`" is an appropriately encoded resource path, which is the identifier of a resource in the Gfdnavi web services; here, the encoding is done by percent-encoding [9].

3.2 Resource Path

The path of a static resource is simply its node path in the directory tree. The path of a dynamic resource is `/{orig_resource}/{operation}`, where "`orig_resource`" is the resource path of original data and "`operation`" is the operation to be performed.

Currently, three types of operations are performed: search, analysis, and visualization. The search `operation` is `find({all_or_first};{queries};{params})`. If `all_or_first` is "`all`", all the search results are returned, whereas if it is "`first`", only the first result is returned. "`Queries`" are search queries joined with an ampersand. Available queries are shown in Table 2. "`Params`" represents search parameters other than the queries joined with a comma. The currently available parameters are `offset={offset}` and `limit={limit}` to specify offset and number of search results to be returned, respectively. Results for searching are sorted by descending order, a parameter to specify the order will be available.

The currently available parameters for analysis and visualization are `analysis({method};{arguments})` and `plot({method};{arguments})`, respectively. Here, "`method`" is the node path of analysis functions or visualization methods, such as `/usr/`

Table 2. Query types for search.

search type	query syntax
keyword search	`kw.{attribution_name}={attribution_value}`
free keyword search	`fw={keyword}`
data type search	`datatype={datatype}`
node path search	`path={path}`
spatial search	`sp.overlap=[{slon},{slat},{elon},{elat}]`
time search	`tm=[{start},{end}]`

`{user_name}/functions/{method_name}` for analysis functions and `/usr/{user_name}/draw_methods/{method_name}` for visualization methods. The methods can be written as `{method_name},{user_name}`. `User_name` and the leading comma can be omitted when `user_name` is `root`. Methods whose `user_name` is `root` are provided by a server administrator and available to all the users in the server, while other methods are added by users.

The original resource could be either a dynamic resource or a static data resource. The resource path for the result of multiple operations is written as a method chain in OOP, such as `object.method1.method2.....` The resource path is written as `/{orig_resource}/{operation1}/{operation2}/....`

An array of resources is also a resource, and its path is `[{resource1},{resource2},...]`. Though every operation can take only one original data resource, the array resource can be used for operations that require multiple data, e.g., `[{resource1},{resource2}]/{operation}`. Further, every operation returns an array resource, and `[{indexes}]` is available to specific required elements of the array. "`Indexes`" denotes an index number of arrays, beginning from 0 or a list of the index joined with a comma; e.g., `/{resource}/{operation}[0,2]` represents an array resource that consists of the first and third elements of an array result.

4 GfdnaviData

We have developed a Ruby class library, named GfdnaviData, as a web service client. GfdnaviData class is a class whose instance object corresponds to a resource of Gfdnavi web services.

Figure 1 shows an example of Ruby script for server-side analysis with GfdnaviData; it yields four HTTP requests shown in Fig. 2. Methods `#find`, `#analysis`, and `#plot` correspond to `/find()`, `/analysis()`, and `/plot()` in the resource path, respectively, where the leading "`#`" in the method name indicates that the method is an instance method. These methods send an HTTP request to an appropriate URL whose extension is "yml", which requires a result in the YAML format. The reason why the result is in the YAML format and not the XML format is that the YAML format is simpler and easier to handle

in Ruby. The instance method `#to_{type}` acquire a resource in the specified format corresponding to "`type`" by setting the extension of the URL; `#to_html` (html), `#to_xml` (xml), `#to_nc` (nc), `#to_knlge` (knlge), `#to_gphys` (gphys), and `#to_png` (png).

Some methods having similarity to GPhys are provided in GfdnaviData. Naive execution of four arithmetic operations with GfdnaviData is slightly complex. For example, to add objects `dataA` and `dataB`, we must write

```
GfdnaviData[dataA,dataB].analysis("addition")
```

, where the `GfdnaviData[]` class method creates an object corresponding to an array resource. Methods `#+`, `#-`, `#*`, and `#/` are added for convenience and for ensuring similarity with GPhys, and then, we can write "`dataA + dataB`" for the above example.

Basic authorization is used for access control in the Web services. Adding your user name and password to `GfdnaviData.parse` as its arguments, or using `#user` and `#password` methods, you can specify your user name and password, respectively. You might not want to embedding password into a script for security reason. GfdnaviData ask your password to you interactively, if you has not specified your password.

4.1 Delayed Execution and Caching

The generation of dynamic resources in Gfdnavi web services, such as analyzed and visualized resources, could cause serious performance problems. The method chain of OOP requires a return object for every method call, which could result in repetition.

Then, in the example shown in Fig. 1, averaging could be executed three times at the last three method calls. In order to prevent such repetition, we introduced delayed execution and caching in Gfdnavi. Operations such as search, analysis, and visualization are not executed, and only resource information is returned for HTTP requests, except for those requests that really require result data. Such a request requiring result data has the following extensions: `html` and `xml` for search, `nc` and `gphys` for analysis, and `png` for visualization. The `#find`, `#analysis`, and `#plot` methods acquire information about only resources in the YAML format. Consequently, in the case of the above example, averaging is executed only once at the last HTTP request, which requires PNG image data, and results in the execution of both averaging and plotting.

As mentioned above, we also included a caching system in Gfdnavi. Analyzed binary data and drawn image data are cached in the memory or storage and used for later requests for the same data. Their lifetime depends on the frequency of use and time required for generating the data.

5 Use of Multiple Servers

We have provided the network capability of use of multiple servers in Gfdnavi. Gfdnavi web services are used internally in Gfdnavi web applications, and Gfd-

naviData is used in Gfdnavi web services for obtaining remote resources from other Gfdnavi servers.

A request for a resource with a URL of the Gfdnavi web services is converted into a GfdnaviData class instance. This instance would send requests to other Gfdnavi servers for remote resources, which implies that Gfdnavi servers could serve as a web service client during cross-server use (Fig. 4). The flowchart of this process is as follows:

1. The GfdnaviData class instance in a client constructs a URL and sends an HTTP request to the server identified by the URL.
2. In the server, the web service controller creates a GfdnaviData class instance corresponding to the requested resource.
3. If required, the instance repeats the process in step 1 as a web service client.
4. The instance executes necessary operations using local data and the results obtained in step 3.
5. The instance returns an object of the requested resource to the controller.
6. On the basis of the returned object, the controller returns a representation of the requested resource to the requesting instance.
7. This instance in the client creates another GfdnaviData class instance or data object such as a binary or image data object, depending on the response.

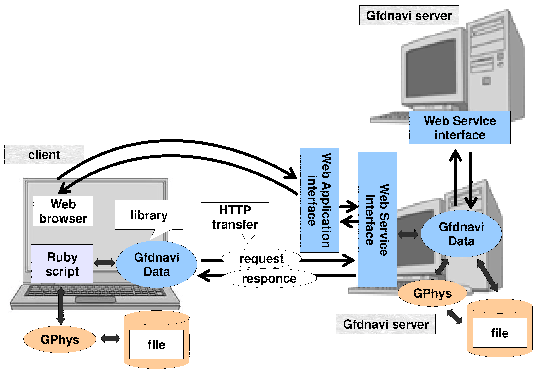

Fig. 4. Structure of web applications and web services provided by Gfdnavi.

5.1 Cross Search

Cross search enables users to search for data on all Gfdnavi servers forming a network for cross search, even if the users know only one server out of them. Cross search has been developed in the Gfdnavi web application using the Gfdnavi web services.

To enable cross search, we have to activate a central server that contains a list of Gfdnavi servers forming a network for cross search. Each Gfdnavi server makes a request to the central server and receives the server list.

Figure 5 shows a screen shot of the cross search page in the Gfdnavi web application. For performing cross search, users select target servers on which search is to be performed.

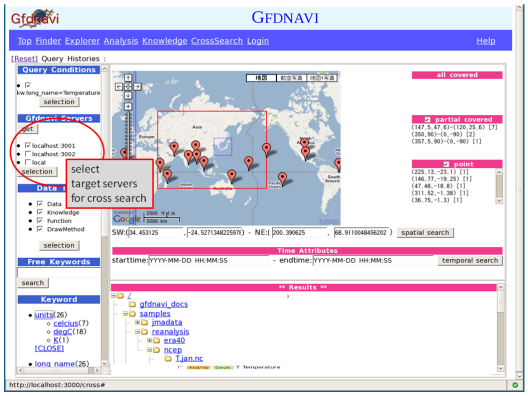

Fig. 5. Screen shot of cross search page in Gfdnavi web application.

The flowchart of the cross search is as follows:

1. A user accesses the search page of a Gfdnavi server, specifies search conditions, and selects those servers from the list that he or she wants to include as search targets.
2. The user can also add other servers that are not in the list to the targets.
3. The search request is sent to the Gfdnavi server.
4. The server executes search for its own data.
5. The server sends the request to the servers specified as search targets via the web services.
6. Then, it collects results from all the specified servers.
7. Finally, it returns the result to the user.

6 Concluding Remarks

In this paper, we have presented the development of the network capability of Gfdnavi. The Gfdnavi web services are developed on the REST architecture. A Ruby client library, termed GfdnaviData, is also developed in order to enable researchers in the field of geophysical fluid sciences to carry out data analysis and visualization on either the server side or the client side and to handle data on multiple servers. A cross search feature across multiple Gfdnavi servers is

realized by using a simple peer-to-peer network with a central server, where the peer-to-peer communication is based on the Gfdnavi web services.

It should be noted that we could not control the reliability and quality of remote data in the cross search. However, it is obvious that in a scientific research, the reliability and quality of data used in analysis are vital. In a conventional way of research, researchers in the field of geophysical fluid sciences directly download data from a known generator or distributor. They usually know the nature and quality of the data that they are working with. In contrast, by using the cross search feature, researchers can acquire unknown data. The data may be of varied quality levels or could be incomplete. In the future, we need to find a way to determine the quality and reliability of data and convey the same to users. At the very least, traceability information of data, such as their generator and distributor, should be made available to users.

Acknowledgments.

We appreciate three anonymous reviewers for their helpful comments. This study was supported by Grant-in-Aid for Scientific Research on Priority Areas "Cyber Infrastructure for the Information-Explosion Era" A01-14 (19024039, 21013002) by the MEXT. We thank Yoshi-Yuki Hayashi, Masato Shiotani, Masaki Ishiwatari, Masatsugu Odaka, and Tomohiro Taniguchi for supporting and promoting this work. We also thank Tsuyoshi Koshiro, Yasuhiro Morikawa, Youhei Sasaki, and Eriko Nishimoto for their valuable contributions and comments.

References

1. Horinouchi, T., Nishizawa, S., Watanabe, C., Tomobayashi, A., Osuka, S., Koshiro, T., GFD Dennou Club: Gfdnavi, Web-based Data and Knowledge Server Software for Geophysical Fluid Sciences, Part I: Rationales, Stand-alone Features, and Supporting Knowledge Documentation Linked to Data In: Data Intensive eScience Workshop 2010, (2010), submitted
2. Gfdnavi, `http://www.gfd-dennou.org/arch/davis/gfdnavi/`
3. Ruby, `http://www.ruby-lang.org/`
4. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine (2000)
5. SOAP, `http://www.w3.org/TR/soap/`
6. Horinouchi, T., Mizuta, R., Nishizawa, S., Tsukahara, D., Takehiro, S.: GPhys - A Multi-purpose Class to Handle Gridded Physical Quantities, `http://ruby.gfd-dennu.org/products/gphys/` (2003)
7. YAML, `http://www.yaml.org`
8. Rew, R., Davis G.: NetCDF – An interface for Scientific-data Access. IEEE Computer Graphics and Applications. 10(4), 76–82 (1990)
9. RFC3986, `http://tools.ietf.org/html/rfc3986`