iEMSs 2008: International Congress on Environmental Modelling and Software
Integrating Sciences and Information Technology for Environmental Assessment and Decision Making
4th Biennial Meeting of iEMSs, http://www.iemss.org/iemss2008/index.php?n=Main.Proceedings
M. Sànchez-Marrè, J. Béjar, J. Comas, A. Rizzoli and G. Guariso (Eds.)
International Environmental Modelling and Software Society (iEMSs), 2008

Using SOAP to Develop Software for Air Trajectory Calculation

J. Zeng[a], T. Matsunaga[a] , H. Mukai

[a] National Institute for Environmental Studies, Onogawa 16-2, Tsukuba, Ibaraki, 305-8506 Japan
Corresponding author: zeng@nies.go.jp

Abstract: We developed a program with Simple Object Access Protocol and constructed a data server system to deliver meteorological data from the server to clients of the program for air trajectory calculation. Our practice shows that with proper design, the performance of running the program with client-server configuration is well acceptable comparing with a program of traditional design. The request-for-data-on-demand approach helps to reduce the burden of maintaining huge amount of data in local machine by end-user and provides a more transparent way to calculate air trajectories.

Keywords*: SOAP; Air Trajectory; Meteorology; Network Computing.*

1. INTRODUCTION

Software for air trajectory calculation is a popular tool for studying airflow pattern [Miller, 1981; Harris, 1992] and source-receptor relation [e.g., Ashbaugh, 1983; Stohl, 1996; Tsuang et al., 2002] in environmental research, especially in pollution or greenhouse gas monitoring [e.g., Moody and Galloway, 1988; Mukai and Suzuki, 1996; Lam et al., 2001]. However, users of such an application often face the difficulties of maintaining huge amount of meteorological data and feeding data to the model of the program. The first problem seems disappearing with the unprecedented increase of storage capacity of personal computers. But new computer power often leads to refining of meteorological models and thus the output volume from models. As a result, there seems never enough storage space for data. Even when storage space is not a problem, downloading GB to TB of data can be time consuming. Meteorological data are subject to changes when new methods for improving data quality become available; therefore updating data is necessary from time to time, which adds additional burden to data maintenance. The second problem comes from the fact that software for air trajectory calculation is usually designed to use data in a specific format. For example, FLEXTRA [Stohl, 1999] is designed for data in GRIB (http://dss.ucar.edu/docs/formats/grib/gribdoc/) format on sigma levels and HYSPLIT [Draxler and Hess, 2004] only accepts data on hybrid levels. Modifying such software for a differently formatted dataset or converting one format to another requires advanced programming skills that often are out of the expertise of researchers. Therefore, a better way to design such software is to make the input transparent by separating the input component from the modelling component.

This article introduces the development of METeorological data EXplorer or METEX (http://db.cger.nies.go.jp/metex/) and the techniques used to solve the problems of data storage and format. By connecting to a data server, the program can be used to calculate air trajectories of several decades in batch mode without storing meteorological data locally. Users do not have to know the format. The server is responsible for format conversion. For advanced users, they can use a scripting language to initialise calculation, to control program execution, and to make new input interfaces for data in different formats.

2. SOFTWARE AND SYSTEM DESIGN

2.1 SOAP Interface

To help reducing the burden of maintaining huge amount of meteorological data in local machine by end-user, we have chosen Simple Object Access Protocol or SOAP (http://www.w3.org/TR/soap/) for sending necessary data from server to client for trajectory calculation. SOAP is a protocol for exchanging XML-based messages over computer networks, normally using HTTP/HTTPS. Among many implementations of SOAP, we found that gSoap (http://www.cs.fsu.edu/~engelen/soap.html) suits our purpose best. The tool generates C/C++ code for SOAP/XML web services from C/C++ header file. The generated code also includes interfaces for serializing both primitive and structured types of C/C++ variable.

Generality often leads to complexity in software design. To balance the two opposite factors, we decided to implement SOAP interfaces for five types of C/C++ variable:

- 32-bits integer for data of absolute precision;
- 64-bits float for number with decimal points;
- Character string for descriptive information;
- Binary array for data array;
- Base64 encoded binary array.

The last two are generic arrays that can be used to transfer binary array of any type. The server and client have to predefine rules for what an array contains. The first array type was used in the early development of METEX and is kept for backward compatibility; and the second was introduced later to reduce the package size of serialized data array. The package size of serialized data produced by gSoap generated C/C++ code for the first array type is several times larger than the source data array. Using base64 encoding, serialized data size is not much larger than 4/3 of the original size.

The C/C++ header for generating SOAP interface for METEX by gSoap is listed in the appendix. Implementation details of the interface can be found in the C++ source code http://www.zegraph.com/z-script/src/metex2.cpp. Here we only outline two key points.

The client component makes request by calling the following C/C++ function generated by gSoap:

```
int soap_call_metex__Request(struct soap *soap, const char
*soap_endpoint, const char *soap_action, char *service, struct
metex__params *params, struct metex__output &r)
```

Which returns response status for the function parameters of

1. *soap* – gSoap object;
2. *soap_endpoint* – server URL;
3. *soap_action* – unused by this implementation;
4. *service* – used by this application to identify request type;
5. *params* – variable containing request parameters whose types are defined in the appendex;
6. *r* – results from server whose types are defined in the appendex.

The server component must implement this C/C++ function:

```
int metex__Request(struct soap *soap, char *service, struct
metex__params* params, struct metex__output &r)
```

Which will be called by the function that waits for client requests. The implementation fills the output variable *r* with data according the contents of *service* and *params* and return proper response status.

2.2 Input and Output Interfaces

A typical air trajectory calculation requires several meteorological variables in two- and three-dimensional, e.g., surface pressure, wind velocities, temperature, and etc. The data size could be in the order of tens of megabytes to hundreds of megabytes depending on the spatial and time resolutions of a dataset and the trajectory length. Hundreds of gigabytes to terabytes of storage space are needed for a research that uses historical meteorological data of several decades. Even if storage space is not a problem, downloading data can be time consuming. Meteorological data are subject to changes when new methods for improving data quality become available; therefore it is necessary to update data from time to time, which adds additional burden to data maintenance.

We identified that in practice most researches involving air trajectory are interested in regional problems and that in such cases only a small portion of meteorological data is needed for trajectory calculation, i.e., data at those grids along a trajectory. Figure 1 shows a 5-day back trajectory calculated for the greenhouse gas monitoring station at Hateruma Island, Japan. Because the uncertainty induced by meteorological data accumulates with the length of trajectory, which is equivalent to the length of weather prediction, much longer trajectories are considered not useful for the research of source-receptor relation. In addition, it does not take much longer to compute a trajectory from one dataset time to another than to read input data with moderate horizontal resolution of 1x1 degree and vertical levels of about 20.

Based on this analysis, we designed the input interface to get data grid-by-grid, instead of data of the whole 2-dimensional or 3-dimentional field. We further separated the IO interfaces of METEX from its trajectory models to maximize the flexibility. The program requests for data through a user object, which in turn tries to read data from a local machine or fetch data from a remote data server. For the output interface, the program either saves results in every hour or sends them to the user object for processing.

Because the modelling component does not have to know the format of input data, the program can be used with any type of data without re-compiling source code as long as an input interface is implemented. So far we have prepared input interface components for data from European Centre for Medium-Range Weather Forecasts (http://www.ecmwf.int/), National Centers for Environmental Prediction (http://www.ncep.noaa.gov/), and Japan Meteorological Agency (http://www.jma.go.jp/jma/indexe.html).

At initialisation, METEX requests for grid sizes and types for each dimension of a given dataset. During trajectory calculation, it requests for data at grids surrounding the current position of air parcel. This approach significantly reduces data traffic through the network.

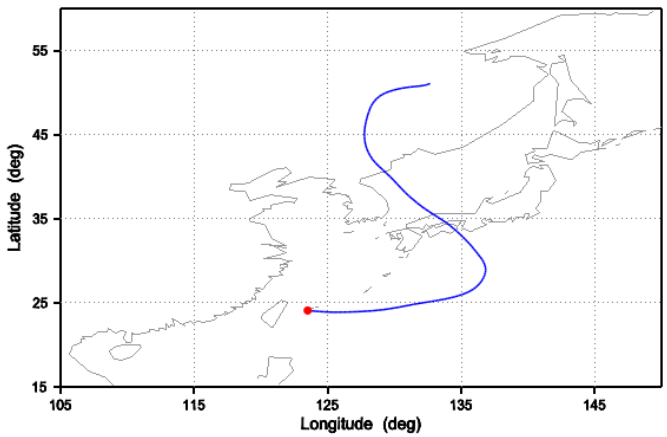

Figure 1. 5-day back trajectory initialised at Hateruma station (123º3'E, 24º49'N) with altitude of 500 m. The initial time is 0:00 UTC, January 1, 2007.

2.3 METEX as Software Library

In contrast to traditional design of environmental software that read initial and control parameters from initial files, METEX is designed as a dynamic link library to be used with a scripting language. The combination makes it even more flexible to handle input and output. The scripting language ZeScript (https://sourceforge.net/projects/zescript/) is used to control every aspect of METEX execution. The C/C++ code generated from gSoap is also bound with ZeScript to run as a service to supply data to METEX. Details for using the library can be found at http://db.cger.nies.go.jp/metex/interfaces.html.

2.4 Data Server

A data server has been set up at the Centre for Global Environmental Research, National Institute for Environmental Studies, Japan. When the URL http://db.cger.nies.go.jp/metex/soap/ is set to METEX, as shown in Figure 2, the program will get necessary data from the server for trajectory calculation. The server has data from January 1, 1950 to at least the previous month of the current date and updates its database from NCEP FTP server (ftp://ftp.cdc.noaa.gov/pub/Datasets/ncep.reanalysis/) at lest once per month.

Users can also download data from the server to their machine to do calculation when the number of calculation is large and the computing speed is critical to their applications.

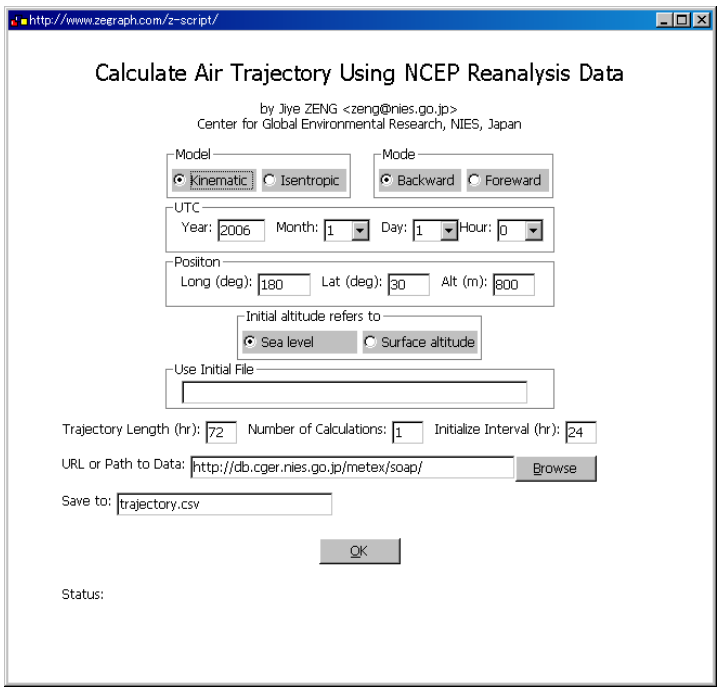

Figure 2. A user interface of METEX. The URL of a data may be set for fetching input data through the Internet. Alternatively, a path to data in local machine can also be set for reading input data locally.

2.5 Model Validation

In addition to visual and statistical comparisons of METEX results with those of other programs, we analysed the correlation between observed concentration change of methane in given time intervals and the end-point distances of backward trajectories that were initialised at observation times. The basic assumption is that the larger the distance, the higher the possibility that sampled air masses came from different regions with different source strength for methane. The results [see Zeng et al., 2003] show that although

trajectory quality varies with initial conditions and model assumptions, they reveal good source-receptor relations in general.

2.6 Test of Server Stability and Computing Speed

We have tested the reliability and stability of the server-client configuration with continuous executions of METEX for input data of January 1950 to December 2006. After solved some troubles at the initial stage, the server is now serving users all over the world. It sometimes accepted over 3 millions requests for data per day. No trouble has occurred in the past three months.

In a series of tests, we initialised the model every 6 hours, which is the time resolution of meteorological data, and calculated 3-day back trajectories over 56 years. The PC used to run tests has 3 GHz CPU, 1 GB memory, and Windows XP; and was located in the same LAN segment of the data server. It took about six days to calculate more than 80,000 trajectories when only one METEX was using the server. The speed is equivalent to 10 trajectory lines per minute. In comparison, the first version of METEX, which was designed in the traditional way to read input data of all grids at once, can calculate about 150 lines of trajectories of the same length with the same PC in one minute. Several similar tests for the server-client configuration with a home PC, located in the same city, shown twice slower speeds at the best condition.

3. CONCLUSION

Network computing used to be the territory of large and complex modelling and simulation. The ever-increasing accessibility and speed of Internet open a new way to develop environmental software for individual user that involves huge amount of data. Taking the advantage of Internet, we used SOAP to add an additional input interface to METEX so that users have the option of either reading data from local machine or request for data from a remote server for air trajectory calculation.

Although the computing speed of METEX running with the client-server configuration is a order of magnitude slower than running with data input from local machine, the performance is well acceptable because in most researches, such a software is used to calculate trajectory for a short period of time. For example, a laboratory in our institute uses it about once per month to obtain trajectories for air sample positions along ship routes; and a municipal government in Japan uses it one per month to archive trajectories for their monitoring stations. The request-for-data-on-demand approach helps to reduce their burden of maintaining meteorological data in local machine because downloading large amount of data can be time consuming and archived data needs updating when the data provider makes changes to improve data quality. With the client-server configuration, the separation of input component from modelling component makes the program transparent and easy to use for end-users.

ACKNOWLEDGMENTS

Dr. Robert A. van Engelen, the developer of gSoap, generously offered his help to use the code generator. This has greatly reduced the time to implement SOAP in METEX.

Appendix. C/C++ Variable Types for Serialization by gSoap

```
enum type_enum { METEX_ANY, METEX_INTEGER, METEX_REAL, METEX_STRING,
METEX_BINARY, METEX_BASE64 };

class metex__any
{ public:
  struct soap *soap;
  metex__any();
  virtual ~metex__any();
  virtual int type() const;
};
```

```
class metex__integer : public metex__any
{ public:
  int value;
  metex__integer();
  virtual ~metex__integer();
  virtual int type() const;
};

class metex__real : public metex__any
{ public:
  double value;
  metex__real();
  virtual ~metex__real();
  virtual int type() const;
};

class metex__string : public metex__any
{ public:
  char *__ptr;
  metex__string();
  virtual ~metex__string();
  virtual int type() const;
};

class metex__binary : public metex__any
{ public:
  int __size;
  unsigned char *__ptr;
  char *msg;
  metex__binary();
  virtual ~metex__binary();
  virtual int type() const;
};

class metex__base64 : public metex__any
{ public:
  char *__ptr;
  char *msg;
  metex__base64();
  virtual ~metex__base64();
  virtual int type() const;
};

struct metex__params{ int __size; metex__any **__ptr; };

struct metex__output{ metex__any *__ptr; };

int metex__Request(char *service, struct metex__params *params, struct
metex__output &r);
```

REFERENCES

Ashbaugh, L.L., A Statistical Trajectory Technique for Determining Air Pollution Source Regions, Journal of the Air Pollution Control Association, 33, 1096-1098, 1983.

Draxler, R.R. and Hess, G.D., Description of the Hysplit_4 Modeling System, NOAA Technical Memorandum ERL ARL-224, 25 pp, 2004.

Harris, J.M., Analysis of Day Midtropospheric Flow Patterns for the South Pole: 1985-1989, Tellus, 44, 409-412, 1992.

Lam, K.S., Wang, T.J., Chan, L.Y., Wang, T., and Harris, J., Flow Patterns Influencing the Seasonal Behaviour of Surface Ozone and Carbon Monoxide at a Coastal Site Near Hong Kong, Atmospheric Environment, 35, 3121-3135, 2001.

Miller, J.M., A Five-Year Climatology of Back Trajectories From the Mauna Loa Observatory, Hawaii, Atmospheric Environment, 15, 1553-1558, 1981.

Moody, J.L. and Galloway, J.N., Quantifying the Relationships Between Atmospheric Transport and the Chemical Composition on Bermuda, Tellus, 40, 463-497, 1988.

Mukai, H. and Suzuki, M., Using Air Trajectory to Analyse the Seasonal Variation of Aerosol Transport to the Oki Island, Atmospheric Environment, 30, 3971-3934, 1996.

Stohl, A., Trajectory Statistics – a New Method to Establish Source-Receptor Relationships of Air Pollutants and Its Application to the Transport of Particulate Sulphate in Europe. Atmospheric Environment, 30, 579-581, 1996.

Stohl, A., The FLEXTRA Trajectory Model Version 3.0, User's Guide, 41 pp, 1999.

Tsuang, B.J., Chen, C.L., Pan, R.C., and Liu, J.H., Quantification on Source/Receptor Relationship of Primary Pollutants and Secondary Aerosols From Ground Sources, Part I. Theory, Atmospheric Environment, 36, 411-419, 2002.

Zeng, J., Tohjima, Y., Fujinuma, Y., and Mukai, H., and Katsumoto, M., A Study of Trajectory Quality Using Methane Measurements from Hatetuma Island, Atmospheric Environment, 37, 1911-1919, 2003.