

# Distributed Application Framework for Earth Science Data Processing

Petr Votava, Rama Nemani, Chad Bowker, Andrew Michaelis,  
Andrew Neuschwander

Numerical Terradynamic Simulation Group, University of Montana, Missoula, MT 59812

Joseph Coughlan

NASA Ames Research Center, Moffett Field, CA 94035

**Abstract** – One of the characteristics of Earth Science data is their diversity, which results in a large number of similar algorithms tailored to a specific data set. Moreover, it is often difficult to connect several algorithms in a “pipeline” where output of one is the input of the other. The solution we propose in this paper is a flexible and an extensible framework that enables us to decouple the data from the application and lay the foundation for integration of several algorithms into a single system.

## I. INTRODUCTION

One of the characteristics of earth science data is their diversity, which results in a large number of similar algorithms tailored to a specific data set. This approach is not very efficient, yet it is often deployed in both small and large projects. Additionally, it is often hard to integrate dependent algorithms in which output of one of the algorithms is input of the next, especially if these were not developed by the same team. Our design addresses these issues by providing a framework for integration of dependent earth science data processing algorithms and for their generalization.

An important feature of the application framework is the decoupling of the data from the algorithms using metadata descriptions. Since the input data can come in many different formats, we are developing a common set of FGDC-compliant [1] metadata to provide a standard description of the data content, including projection, data types, fill values, etc. Metadata is stored in XML [2] format, and the data we use internally is in HDF5 [3] format. Rather than rewriting the data processing algorithms to fit the appropriate data set, we implement filters that preprocess the data to the format required by the algorithms. This facilitates faster development and encourages code re-use.

Many Earth science algorithms are very complex, but they also often have only a small degree of spatial dependency and thus are ideal for parallel processing. We utilize the distributed features of the Java [4] programming language to accommodate parallel processing. With our framework we can build flexible and scalable processing “pipelines” that include preprocessing, processing and automated result analysis as independent modules. This gives us the flexibility to add and remove modules on the fly, as well as re-use

existing code, and thus enables us to concentrate more on the science itself rather than on system integration. Finally, we implement a batch mode so that the system can run without user interventions for long periods of time, providing the scientists with an automated way to obtain the results they need quickly and efficiently.

The remainder of the paper is organized as follows. Section 2 provides information on the data description scheme. Section 3 describes the distributed system architecture. Section 4 gives 2 examples of current deployment of the framework in our TOPS [5] and MODISWeb [6] systems. We conclude with a discussion of future work.

## II. DATA DESCRIPTIONS

In order for an application to be able to handle multiple data formats in a flexible way, it needs to obtain detailed information about the data – this information can range from data type to distribution information. Because the data vary so greatly in their formats, from ASCII and simple binary, to Hierarchical Data Format (HDF) and HDF-EOS, we had to find a metadata scheme that would be capable of including all the different datasets that are of interest to the Earth science community. We have decided to use metadata standard developed by the Federal Geographic Data Committee (FGDC) [1]. The standard specifies a generic framework that can be used to describe any geospatial data with regard to the following aspects: Identification, Data Quality, Spatial Data Organization, Spatial Reference, Entity and Attribute Information, Distribution, and Metadata Reference. The standard also specifies which of the above sections are mandatory and which are optional. This simplifies greatly the data descriptions in case of simple data sets when not all the information has to be included, but remains very expressive in description of complex data sets. FGDC also provides a set of tools for checking that metadata conforms to the specifications, and for conversion to XML [2] and HTML [7] formats.

Since main parts of our application framework are written in Java, we have decided to use XML for the metadata implementation, because Java provides extensive support for

handling of XML documents. It is the Java and XML combination that brings the flexibility and extensibility into the design of the application framework.

Fig. 1. shows the situation where application is not aware of the format of the input data. It simply makes a request to the Data Broker object with regard to the type of the data and its location, and it receives the data. Where the data comes from depends on the particular inputs available to the system at that time. This is very useful for evaluating and testing of new algorithms and data streams, where we want to be able to use the same algorithm with two different input data streams without having to deal with the different data formats at the science algorithm level. This provides for faster algorithm development cycle, and the ability of the scientist to concentrate more on the science of the algorithm, rather than input data formats. One thing to notice is that FGDC metadata standard also addresses the aspect of the quality of the data, so the application has an access not only to the data itself, but also to its QA information that at times is required in Earth science algorithms. An example QA usage is information whether a particular pixel contained any clouds.

Because the Data Broker knows all the relevant information about the data, it is able not only to obtain the data, but also to perform simple transformations. Examples of such transformations are reprojection, subsetting, or resolution adjustments; these can be specified as additional parameters during calls to the Data Broker object.

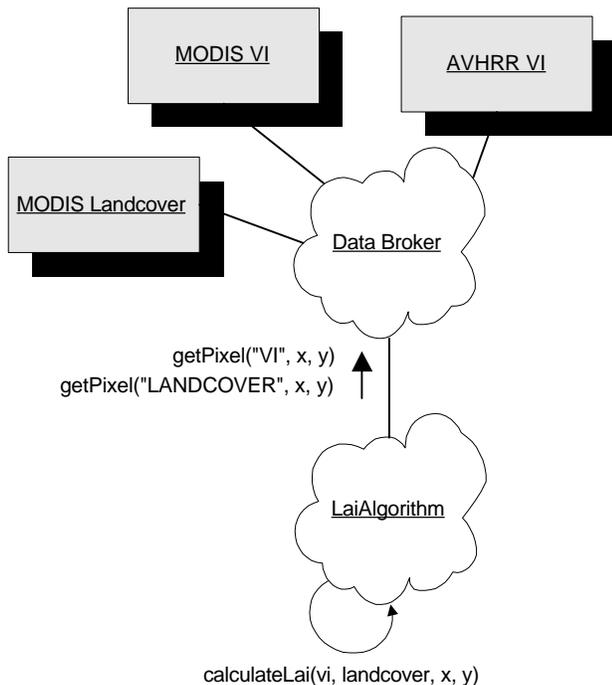


Fig. 1. Application independent of its input data representation

There are two types of data that the framework must be able to handle – *external* that are coming from outside of the system, and *internal* that are produced by the system. While we have little control over the external data and their formats besides creating the FGDC-compliant metadata descriptions, we have decided to use HDF5 [3] for internal data representation. HDF5 is the latest version of the Hierarchical Data Format that provides better support for Java and new internal organization that is very suitable for our applications.

### III. DISTRIBUTED FRAMEWORK ARCHITECTURE

One of the many aspects of the Earth science data processing is the volume of the data. This fact together with time complexity of some of the algorithms led us to provide mechanisms for parallel execution of the data processing whenever possible. For example, a task of creating images of continental US from MODIS [8] data involves reprojection, mosaicing, subsampling and image conversion of the data. The MODIS data comes in the form of tiles (continental US consists of about 20 tiles) that, at least during part of the processing, are independent of each other. The part of the system that handles the parallelism is the scheduler. The user/developer submits a request to the scheduler describing what he/she wants to accomplish and how, and the scheduler will load appropriate algorithm modules, I/O modules, and setup the execution sequence that corresponds to the user’s requirements, executing modules in parallel whenever possible. In the example above, the scheduler would execute all the reprojection processes in parallel with synchronization point before entering the mosaicing process. Fig. 2. illustrates this process.

The implementation of the execution environment is done by facilities of Java RMI [9]. Each algorithm object has to implement a *JobInterface*, and provide the code for *execute()* method of the class. This object is then passed to the scheduler, which in turn forwards it to one of the execution servers. The execution server calls the *execute()* method provided by the object and returns the results of the execution to the caller. The algorithms themselves are often implemented in C or C++ and we use the Java Native Interface (JNI) [10] to make calls to the shared libraries that contain the appropriate modules.

### IV. TOPS AND MODISWeb EXAMPLES

We are currently using our application framework on two different systems. First, there is the Terrestrial Observation and Prediction System (TOPS) [5]. TOPS is a very good case study for this framework, because it consists of large number of differently formatted inputs (HDF-EOS, wgrib, ASCII, binary) and it requires several algorithms to be run on different inputs before they can be used by the main compute

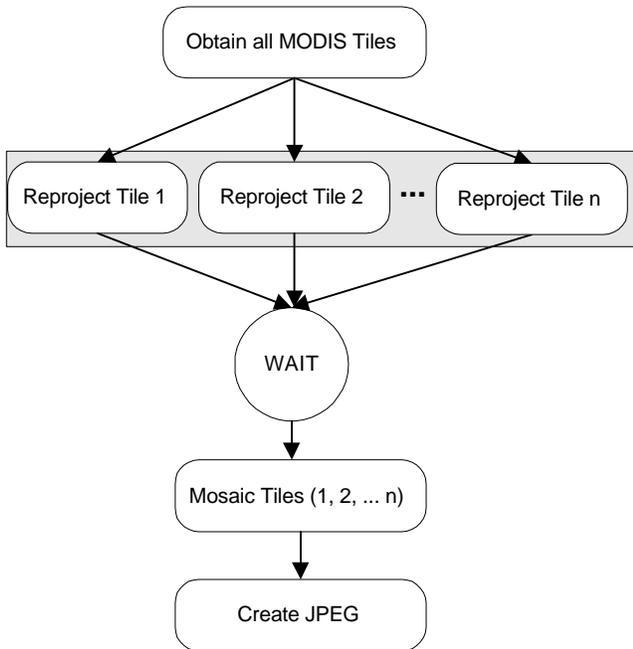


Fig. 2. Parallel processing of MODIS tiles in MODISWeb

engine. At the writing of this paper parts of TOPS are still in experimental stage, but intermediate results suggest that the flexibility in the application framework will enable us to use TOPS for near-real-time forecasting with failover input feeds.

Second project that uses the application framework is the MODISWeb, which is an automated image utility. MODIS data are processed for 17 different scenes around the world in 3 different resolutions and posted on the project Web site. The MODIS tiles for each of the scenes are reprojected, mosaiced, subsetting, and converted to a JPEG image. As the last step, the process updates a database that is used to dynamically load the created images by the Web server. This project is currently in beta stage and the resulting images are being posted to <http://images.ntsug.umt.edu>.

## V. FUTURE WORK

We are currently adding functionality into the framework to support improved feedback and analysis of results through data mining and machine learning. This will help us to dynamically improve some of our forecasting models by evaluating the results of our forecasts against real data. We are also adding a planner that will help the scheduler decide on the sequence of actions based on specified goals. Finally, we are starting to design a natural language interface as the front end of the framework that would enable us to query the system with questions of the type: "What is the flood danger for Missoula valley in May 2002?"

## ACKNOWLEDGMENT

The Distributed Application Framework project has been funded by NASA IDU program.

## REFERENCES

- [1] Federal Geographic Data Committee, "Content standard for digital geospatial metadata workbook version 2.0", FGDC, May 2000.
- [2] World Wide Web Consortium, "XML: Extensible Markup Language", W3C, <http://www.w3.org/XML>
- [3] The National Center for Supercomputing Applications, "HDF5 abstract data model", presented to NASA EOS/HDF Workshop, September 1999.
- [4] K. Arnold, J. Gosling and D. Holmes, *The Java Programming Language Third Edition*, Palo Alto, CA: Addison Wesley, 2000.
- [5] R. R. Nemani., M.A. White, P. Votava, J. Glassy, J. Roads and S.W. Running, "Biospheric forecasting system for natural resource management", *Proceedings of the 4<sup>th</sup> International Conference on Integrating GIS and Environmental Modeling (GIS/EM4): Problems, Prospects and Research Needs*, Banff, Alberta, Canada, September 2-8, 2000, pg44-52.
- [6] MODISWeb, <http://images.ntsug.umt.edu>
- [7] World Wide Web Consortium, "Hypertext Markup Language", W3C, <http://www.w3.org/MarkUp>
- [8] C. Justice at al., "The Moderate Resolution Imaging Spectroradiometer (MODIS): Land remote sensing for global change research", *IEEE Transactions on Geoscience and Remote Sensing*, 36(4), 1228-1249, 1998.
- [9] E.Pitt, K. McNiff, *java.rmi: The Remote Method Invocation Guide*, Addison Wesley, 2001.
- [10] R. Gordon, *Essential JNI: Java Native Interface*, Prentice Hall, 1998.