Graphyte software for integrated remote sensing research using HPCC

Michael D. Grossberg[a], Joesan A. Gabaldon Jr.[a], Paul K. Alabi[a], Jeremy K. Neiman[a] and Irina Gladkova[a]

[a]City College of New York, 160 Convent Avenue, New York, NY 10031

ABSTRACT

The broad goal of GEOSS-V, creating a unified system of systems that encompasses all relevant atmospheric and environmental remote sensing data, accesses social and economic impact information, and integrates all relevant analysis and decision-making tools, is a monumental task. This is made more difficult in that as technology and algorithms change at an ever-increasing pace, the ability to test, prototype, and integrate new technology is often difficult in large production systems. We have been developing Graphyte, a very flexible lightweight integration framework which is aimed at augmenting GEOSS-V technology by providing agile development tools for research and development.

Keywords: GEOSS-V, Software, Systems , Remote Sensing, Social, Economic, Information, Technology, Integration, Analysis, Development, Research, High Performance, Computing,Tools, Algorithm

1. INTRODUCTION

Graphyte is a project to create software which enables a collaborative and distributed environment for research. Rather than creating a new system structure, its goal is to integrate tools, interfaces, and data that scientists are already using to develop scientific scripts and programs. In addition, the system encourages some of the best practices from modern software engineering. It integrates distributed version control, grid and cloud computation, data management, advanced analysis, and visualization tools with multiple clients, including a web-based interface. The first prototype of Graphyte, called "Atlas,"[1] provides a through-the-web interface, which permits the editing of Python scripts to read, statistically analyze, and visualize NOAA's remote sensing data. Based on user feedback, we have completely restructured the system to better accommodate NOAA's current workflows. The new version, codenamed "Betelgeuse," has many new capabilities. In addition to Python, the system is now able to handle virtually any research coding language that the execution environment supports, making it compatible with languages currently used at NOAA such as Fortran, IDL,[2] and C/C++. Execution is managed by a sophisticated job system which is able to use grid and cloud High Performance Computing (HPC) resources. The system also provides for the management and reusability of distributed data.

Another important aspect of Graphyte Betelgeuse is that it provides fine-grained user and group permissions. This allows researchers to select with whom they want to share code, data, and results. This is a critical pre-requisite for any practical adoption in a research context. In addition to the wide and deep functionality provided by accessing open-source libraries within the system, such as for GIS, economic analysis, machine learning, statistics, image processing, physics, and visualization of data, we provide wrappers for convenient access to NOAA and NASA remote sensing data.

Further author information: (Send correspondence to M.G.)
M.G.: E-mail: grossberg@cs.ccny.cuny.edu, Telephone: 212-650-6166

2. BACKGROUND

There are a number of systems that provide a complete workbench of tools for computation and analysis of data. The McIDAS-X[3] and V[3] and ENVI[2] software are desktop applications that combine interactive viewers with built-in image analysis capability. While Graphyte does not currently support interactive analysis, Graphyte clients can provide interactive viewing of results such as a web client with a 3D viewer for 3D data, or pan and zoom viewers for images. In addition, it is possible to interoperate with some of these interactive environments through backend plugins. For example a plugin could be written for the Java-based McIDAS-V, allowing it to run scripts through Graphyte and return results for interactive viewing.

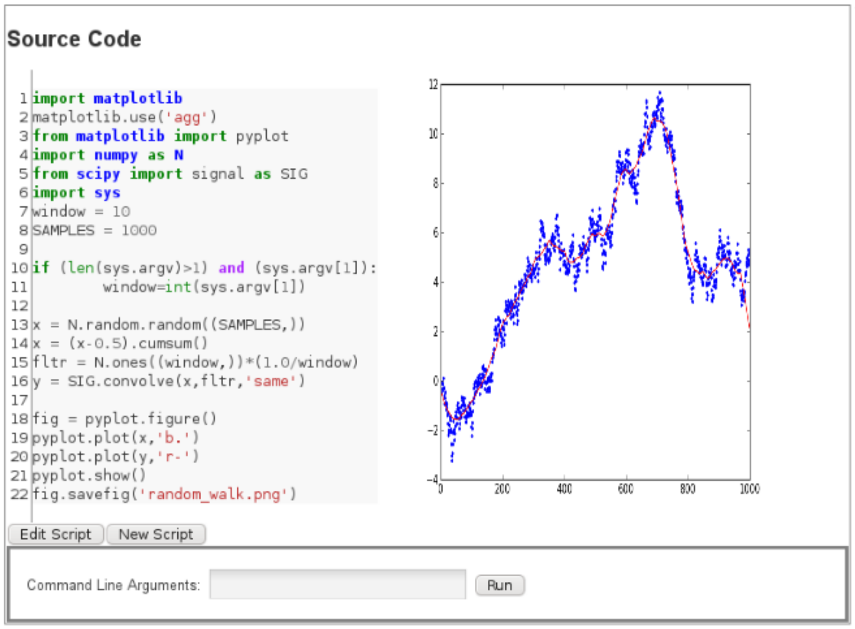
Another way Graphyte can work together with current technology is that many current tools can be scripted. For instance, McIDAS-V, IDL[2] or MATLAB can run scripts which dump output to files in batch mode. Any batch job can be dispatched by Graphyte Betelgeuse to a machine that can run that script, as long as one is available to the job dispatcher with the proper software and licenses. This gives two advantages. One is that it means one can initiate and manage runs through a variety of clients. For example, a web interface could be used to run a McIDAS script from a mobile device, even if McIDAS does not directly provide such an interface. Another advantage is that because outputs for runs are stored in the system, they can be used as inputs for future runs. That makes it possible to link a run using one program environment, say IDL, through outputs, to a run using another, such as Python. While it is possible to do this kind of thing manually from a workstation, it could be run through a web or shell client remotely using Graphyte. Unlike trying to do this directly using a bash shell, data is managed and tagged for the user, and history is kept.

There are other systems that allow scientific workflow processing, though not specifically for remote sensing, such as Vistrails, Taverna, and Kepler.[4–6] In addition, there are web-based workflow systems such as HubZero and MyExperiment.com.[7–9] A number of these tools were surveyed in a paper by Börner proposing Plug-and-Play Macrosocopes.[10] Typically these systems are based on component-based software engineering. Users must componentize there code as plugins or modules, which could then be graphically linked within these systems. While this is a powerful framework that has the potential to unify source code of heterogeneous types into a single coherent architecture, users who are simply looking for a system to simply run a script may not be willing to componentize their code. Graphyte more easily accommodates the user's current code and workflow since it acts more as a script that dispatches the user's code to be run on some system that can accommodate it, rather than attempting to be a universal environment for running code itself. It should be noted, however, that although the current version of Graphyte is not adapted to do so, it would be possible to use a workflow system such as described above as a computational server for Graphyte. Thus, to the extent that the code in the workflow system could be specified as a set of files, Graphyte could store those files, submit them to the workflow system, and retrieve the outputs. In this way it, could add value to a workflow system in that it could make it better interoperate with other non-workflow/non-component based software.

There are other systems that permit editing and execution of code through the web. For instance the Sage project gives the ability to execute programs as scientific notebooks in a large set of scripting languages, including Python, R, and MATLAB.[11] Nevertheless, it is not meant to develop large model runs or to build and execute general-purpose code with multiple files. The cloud service Kodingen lets you edit and run code through the web but it is designed for deploying web applications rather than scientific batches.[12] Again, since Graphyte dispatches jobs to other servers for computation, it would be possible to use these services through Graphyte and thus combine them with other services. In short, Graphyte provides a concierge-like service which brokers batch computation, and tracks and manages data input and output for the users, and through multiple clients. This kind of service is unique.

3. GRAPHYTE ATLAS

Graphyte Atlas is the initial version of the Graphyte toolkit system. A small group of scientists at NOAA and partner institutions are currently testing this system. Atlas provides a web interface to analyze data, execute code, optionally in an HPCC environment, view results, and collaborate remotely. We have previously reported our early development Atlas CITE. By continuing to integrate Python libraries, adding scripts, developing documentation, and wrapping Fortran retrieval code, we have improved the system's functionality.



```python
import matplotlib
matplotlib.use('agg')
from matplotlib import pyplot
import numpy as N
from scipy import signal as SIG
import sys
window = 10
SAMPLES = 1000

if (len(sys.argv)>1) and (sys.argv[1]):
        window=int(sys.argv[1])

x = N.random.random((SAMPLES,))
x = (x-0.5).cumsum()
fltr = N.ones((window,))*(1.0/window)
y = SIG.convolve(x,fltr,'same')

fig = pyplot.figure()
pyplot.plot(x,'b.')
pyplot.plot(y,'r-')
pyplot.show()
fig.savefig('random_walk.png')
```

Figure 1. An example of a python script in Graphyte atlas and the image generated

Currently the system provides accessibility to a wide range of numerical, scientific, and mathematical libraries through Python scripts. For example, figure 1 shows a script to generate a random walk. It was written and run through a web page alongside the generated output. The plot was generated using the Python Matplotlib visualization library. The numerical operations were performed by the Python Numpy library which supports a full suite of basic matrix operations and numerical programming. It is important to note that although called from Python, the mathematical and matrix operations are actually implemented and executed in compiled C or Fortran an thus are quite fast.

The companion library SciPy provides a full range of functions for probability and statistics, Fourier transforms, linear and non-linear optimization, numerical integration, regression, signal processing, data clustering, interpolation, multi-dimensional image processing, and MATLAB data compatibility. In addition, Matplotlib provides data visualization, and basemap provides geographic re-projection of vector maps and wraps the widely used mapping library proj4. More recently, we are integrating the NCAR library PyNGL which provides the ability to create meteorological visualizations, such as Skew-T plots, as shown in Figure 2.

We have included a number of libraries to work with data, as well as having written our own software to display and analyze remote sensing data. There are functions for reading an extensive array of remote sensing data formats including HDF4/5, NetCDF, Envi,[2] NDF, png, GeoTiff, GRIB1, GRIB2, Shapefile, MapInfo, and GMT, as well as generic formats like CSV, Excel, XML, JSON, and Yaml. Figure 3 shows a MODIS granule in HDF that was read in and displayed using Matplotlib, within Graphyte. Figure 4 shows a GOES sounder image that has been remapped using proj4, interpolated using Scipy, then displayed using basemap and Matplotlib. As another example, the GOES sounder data was processed by Python-wrapped Fortran code provided by Ken Pryor to compute the Microburst Wind Potential Index[13] as shown in Figure 5. Using FAA data for airport locations, and the Python library basemap for mapping, Pryor has been using Graphyte for on-demand visualizations of this measure of severe weather risk.

Within Atlas, users are given storage space to which they can upload data of their choosing. They can then access their files from within the system by specifying a path on a shared file system. In addition to that, we have large collections of shared data stores which include data from MODIS, SEVIRI, AVHRR, GOES, and NCEP/NCAR[14] data sets. Further data outside Atlas can be accessed on demand through web data services such as OpenDAP. In addition to accessing data services, it is possible from within ATLAS to access High

Performance Computing Center (HPCC) resources, though in a limited way. We created methods that could be called within a script to log in and send the script to an HPCC.

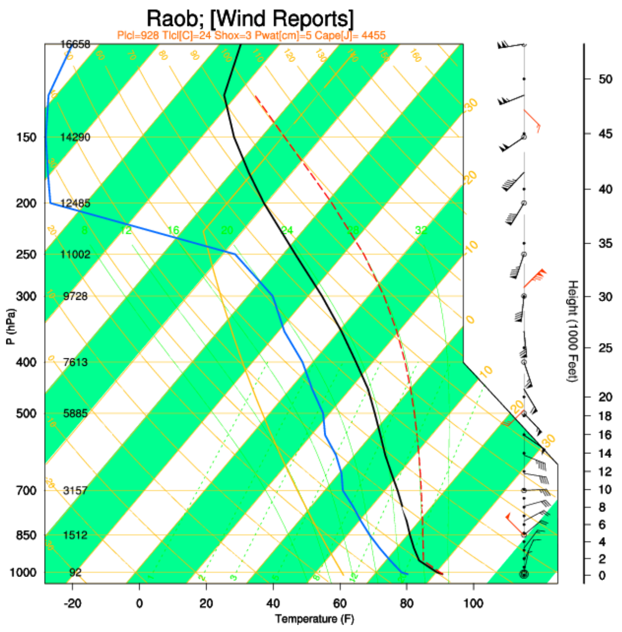


Figure 2. Skew-T plot from the NCAR PyNGL library which can be generated through Graphyte

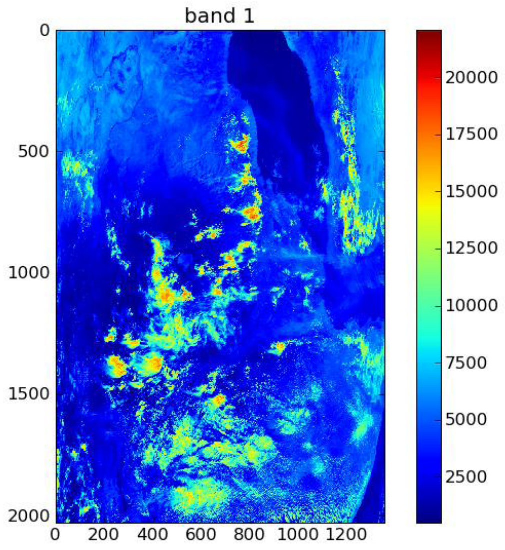


Figure 3. A MODIS image plotted in Graphyte using Matplotlib

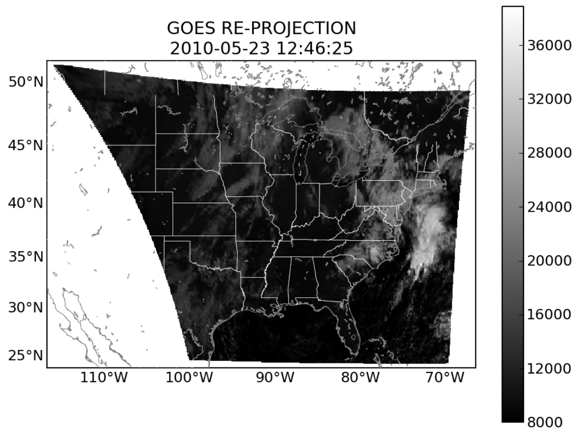


Figure 4. Image generated from python script for reading a GOES Sounder file and remapping it using matplotlib and Basemap.

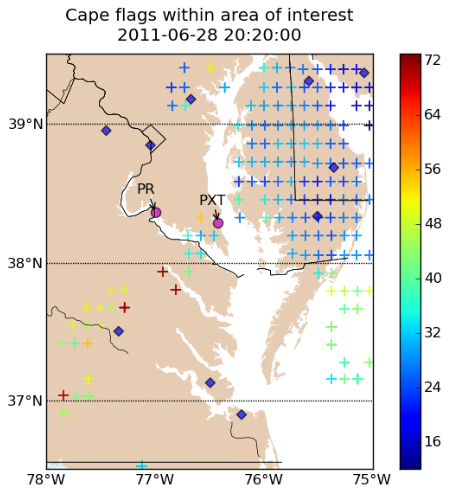


Figure 5. Map generated from python wrapper for fortran code that calculates GOES sounder MWPI above a risk threshold with airports and locations of interest marked.

Graphyte Atlas has provided a demonstration of the potential of such a collaborative system, yet it has been missing some important features. One problem is that the system could only handle single-file scripts. This is a problem both for reusability and for organization. In addition, although the system as a whole was protected by a login, there is no mechanism in Graphyte Atlas to make code private, or share it only with specific people. Another problem is that Atlas does not provide data storage to users beyond uploading to their user accounts. If the user moved the data accidentally, the script would break. Also, the system should transparently leverage

HPCC resources rather than force the user to handle communication with the HPCC within their script. Finally, and perhaps most importantly, we found from interviews with users that it was essential to be able to run code in any language without significant modification. Despite the fact that MATLAB and IDL scripts may be easily translated to Python, and that Fortran, C/C++ and Java may be accessed from Python, some users told us that unless they could directly run their code in these languages, they were unwilling to use the system. To address this we have radically redesigned the system, and call the new version Betelgeuse.

4. GRAPHYTE BETELGEUSE ARCHITECTURE

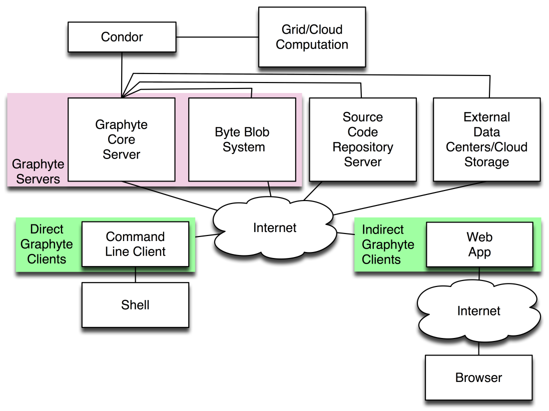


Figure 6. Graphyte Software Architecture Diagram

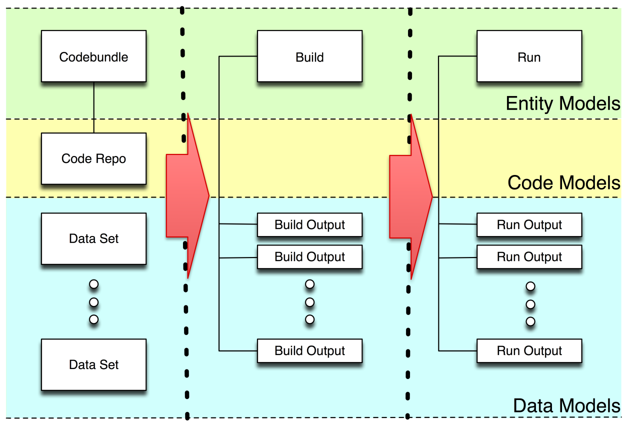


Figure 7. Graphyte System Models

In order to accommodate all the features previously missing, described in the previous section, Graphyte Betelgeuse has been factored into separate client and server parts as shown in figure 6. Rather than being designed around web application, Betelgeuse is designed to support multiple clients, with core functionality being provided as a service to the clients. When a user interacts with Graphyte, the user may interact with direct or indirect clients. The direct clients allow the user to connect to the Graphyte system directly, such as with a shell, but require software to be installed on the user's machine. Indirect clients allow users to access the system without installing any Graphyte-specific software, for example using a web browser.

An important aspect of Graphyte Betelgeuse architecture is that the system delegates much of its functionality to external systems. For example, the source code for Graphyte is stored in an external source code repository. This means that Graphyte can be integrated into current workflow practices. As another example, Graphyte can use a variety of types of external data storage such as disk or cloud storage facilities. Finally, Graphyte can use the Condor dispatcher. Through Condor, computation can occur on local clusters, remote grids, or even using cloud computation. Graphyte acts as a concierge managing a virtual integration of disparate external services. Graphyte clients interact directly with external services whenever possible. However, if an operation requires multiple services to be coordinated, the Graphyte core server (GCS) provides integration through an Application Programming Interface (API) provided to the clients. For an example, when a run is complete, the GCS will move output artifacts from the Condor master to external data storage. The Byte Blob Data System (BBDS) maintains identifiers to all artifacts the system uses, as well as the location in internal or external systems that those artifacts can be retrieved. The BBDS also stores metadata on byte blobs.

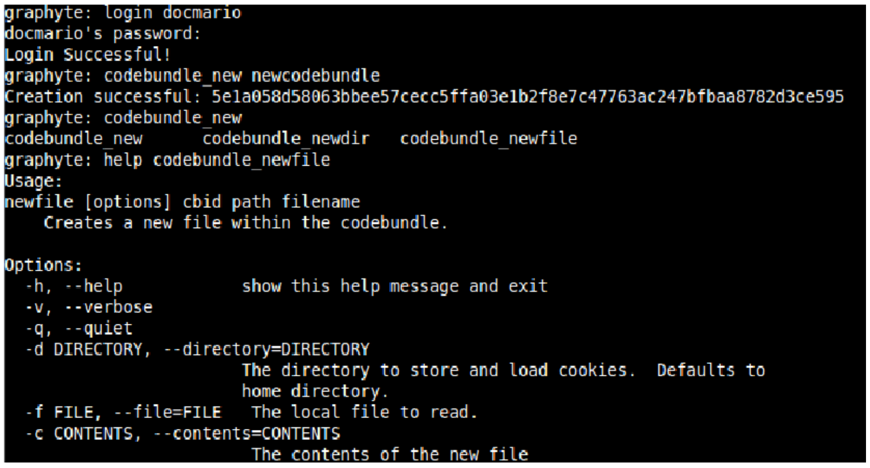
The primary objects of the Graphyte system are represented by a set of models shown in figure 7. A codebundle is a repository of user source code and some metadata that permits the code to be built and run. As noted earlier, the repository itself is not kept within the system. The GCS just stores the information that it needs to perform the required operations on the repository. Data sets, built code, and outputs from previous runs, are all considered data sets, and are tracked by data models in the BBDS. To perform a build, the data is associated with the codebundle through a configuration file. In preparation for the build the data is either linked or copied to the correct location so when the build is executed it is where the code expects it to be in order to accomplish a successful build. The build is a container with metadata. If the build is successful, it refers to a number of build outputs which are output artifacts. Those outputs are tracked by the BBDS system. After a build, the build outputs, along with potentially more linked data and libraries, can be associated in the process of running the build to create a run. Builds and runs are similar except that runs are terminal: they can only be viewed, not modified. A build can be executed many times with different data and parameters, creating many runs. Run outputs, like build outputs, are stored and tracked by the BBDS, with the run model providing a permanent record of the run. Thus these models of codebundle, build, and run, and the operations on them, are the primary focus of Graphyte and are accessible through the various Graphyte clients.

5. GRAPHYTE CLIENTS

Graphyte Betelgeuse is designed so that it is easy to create clients that are adapted to best match a user's needs. All clients use the Graphyte server to mediate the interaction between the external services provided by code repositories, computational servers, and data centers. The clients interact directly with those services whenever communication with the Graphyte servers is unnecessary, such as when a user needs to update code in a source code repository. The user interacts with Graphyte through one of two classes of clients: direct and indirect clients.

5.1 Direct Clients

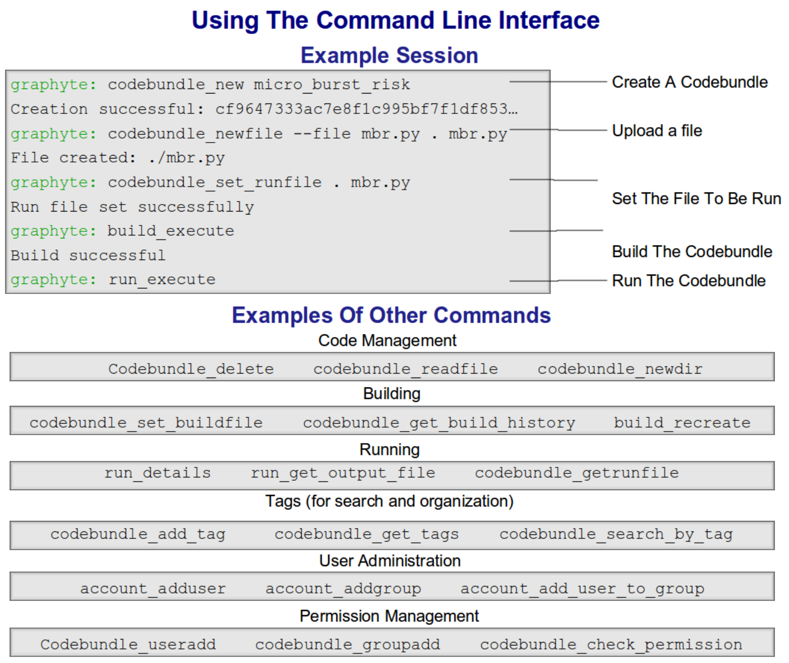
A Graphyte client can be used as a plugin to a more complex system, such as a pre-existing visualization application such as MacIDAS or through a simple shell client. It is also planned to develop native mobile and tablet apps, particularly for viewing data and visualization. For such direct clients, users would need to install Graphyte software locally.



```
graphyte: login docmario
docmario's password:
Login Successful!
graphyte: codebundle_new newcodebundle
Creation successful: 5e1a058d58063bbee57cecc5ffa03e1b2f8e7c47763ac247bfbaa8782d3ce595
graphyte: codebundle_new
codebundle_new      codebundle_newdir   codebundle_newfile
graphyte: help codebundle_newfile
Usage:
newfile [options] cbid path filename
    Creates a new file within the codebundle.

Options:
  -h, --help            show this help message and exit
  -v, --verbose
  -q, --quiet
  -d DIRECTORY, --directory=DIRECTORY
                        The directory to store and load cookies.  Defaults to
                        home directory.
  -f FILE, --file=FILE   The local file to read.
  -c CONTENTS, --contents=CONTENTS
                         The contents of the new file
```

Figure 8. Screen capture of a help message viewed from the GCLI



```
graphyte: codebundle_new micro_burst_risk
Creation successful: cf9647333ac7e8f1c995bf7f1df853…
graphyte: codebundle_newfile --file mbr.py . mbr.py
File created: ./mbr.py
graphyte: codebundle_set_runfile . mbr.py
Run file set successfully
graphyte: build_execute
Build successful
graphyte: run_execute
```

```
Codebundle_delete     codebundle_readfile     codebundle_newdir
```

```
codebundle_set_buildfile     codebundle_get_build_history     build_recreate
```

```
run_details     run_get_output_file     codebundle_getrunfile
```

```
codebundle_add_tag      codebundle_get_tags     codebundle_search_by_tag
```

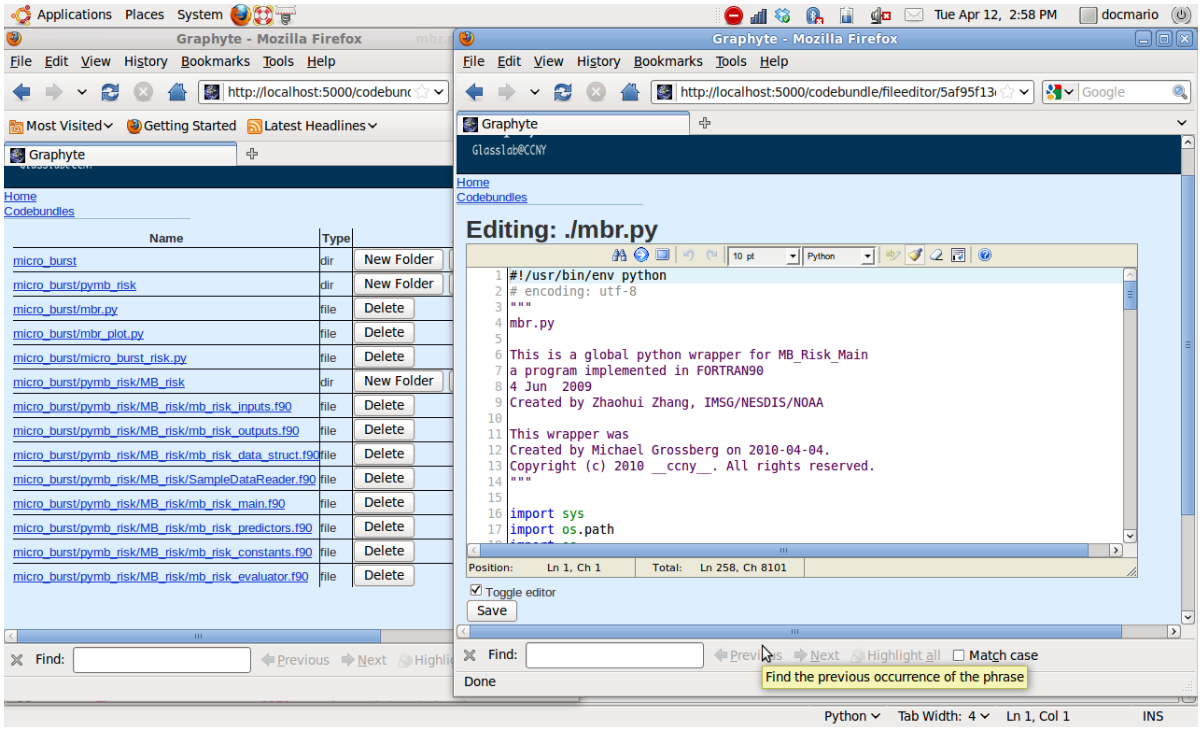
```
account_adduser      account_addgroup     account_add_user_to_group
```

```
Codebundle_useradd     codebundle_groupadd     codebundle_check_permission
```

Figure 9. Commands and example input to the Graphyte Command Line Interface (GCLI)

For instance users can download and run an installer which sets up a shell client, the Graphyte Command Line Interface (GCLI), as shown in figure 8. This shell provides some basic operations for creating Codebundles from scratch, or from an existing repository of code. Some of the commands and sample interactions are shown in figure 9. Users can build and run experiments from this shell. The shell can also be configured to automatically download results from the remote server. The simple workflow of run and build allows users to exploit HPCC resources transparently. The GCLI supports all the operations of Graphyte, but, because it is executed through the shell, the user can easily mix and match with other tools they are currently using. For instance code can be edited in a user's preferred text editor, or data can be explored using a preferred data viewer like McIDAS. While a direct client like the GCLI can provide seamless and flexible integration with a user's current workflow, it still requires software to be installed, and relies on the user's own software for editing and viewing of data.

5.2 Indirect Clients



Name	Type	
micro_burst	dir	New Folder
micro_burst/pymb_risk	dir	New Folder
micro_burst/mbr.py	file	Delete
micro_burst/mbr_plot.py	file	Delete
micro_burst/micro_burst_risk.py	file	Delete
micro_burst/pymb_risk/MB_risk	dir	New Folder
micro_burst/pymb_risk/MB_risk/mb_risk_inputs.f90	file	Delete
micro_burst/pymb_risk/MB_risk/mb_risk_outputs.f90	file	Delete
micro_burst/pymb_risk/MB_risk/mb_risk_data_struct.f90	file	Delete
micro_burst/pymb_risk/MB_risk/SampleDataReader.f90	file	Delete
micro_burst/pymb_risk/MB_risk/mb_risk_main.f90	file	Delete
micro_burst/pymb_risk/MB_risk/mb_risk_predictors.f90	file	Delete
micro_burst/pymb_risk/MB_risk/mb_risk_constants.f90	file	Delete
micro_burst/pymb_risk/MB_risk/mb_risk_evaluator.f90	file	Delete

Editing: ./mbr.py

10 pt Python

```python
#!/usr/bin/env python
# encoding: utf-8
"""
mbr.py

This is a global python wrapper for MB_Risk_Main
a program implemented in FORTRAN90
4 Jun  2009
Created by Zhaohui Zhang, IMSG/NESDIS/NOAA

This wrapper was
Created by Michael Grossberg on 2010-04-04.
Copyright (c) 2010 __ccny__. All rights reserved.
"""

import sys
import os.path
```

Position: Ln 1, Ch 1 Total: Ln 258, Ch 8101

☑ Toggle editor

Save

Figure 10. Screen capture of the web interface showing the browsing listing of Python and Fortran files, along with an editor view of one file.

In contrast to the GCLI and direct Graphyte clients, an indirect Graphyte client is a client that allows users to interact with the system through pre-existing software for communicating with internet services such as email, instant messaging and the web. For instance, one Graphyte indirect client is a web application. In this case the Graphyte client actually runs through a web server. The user's actual client is an ordinary web browser. This means no new software need be installed on the user's machine if they have a web browser.

The web client provides access to all the operations of a scientific workflow: editing files, building code, attaching data, and running an experiment. The web browser has an embedded editor with syntax highlighting, and outputs are viewed in the web application through viewers or in the browser directly. These features were also available in Graphyte Atlas. The new features are the ability to work with a directory containing multiple files as shown in figure 10. In addition, as with the GCLI, the system can handle a wide range of programming languages such as MATLAB, IDL, Fortran, C/C++, Python, and Java through this web interface. The only restriction is that it must be able to be run as a batch process, and the computational server (external to Graphyte) must be configured to run it. Further, the web client provides access to code repository operations such as clone and merge. This makes it even easier to develop code for experimentation using software development best practices.

6. CODEBUNDLES AND SOURCE CODE REPOSITORIES

The codebundle model from figure 7 is an object that holds, or references, the information needed to build and run the experiment. The codebundle contains a reference to a code repository (repo) where all of the source code is kept. It also contains meta information, such as specifying which file in the repo is responsible for building it (e.g., a makefile). Other information that controls how the code is built, for example system requirements on the build environment or data needed to create a build, can be specified in the codebundle.

The repo referenced in the codebundle is not formally part of Graphyte. Graphyte manages storing the code in the code repository for the user, but the repo is hosted on a separate server and could, in theory, be on a public collection of repositories such as Souceforge, Bitbucket or Github. We currently use scm-manager to hold repositories for Graphyte. Scm-manager supports svn, Mercurial, and git. At the moment, Graphyte Betelgeuse works only with Mercurial but will soon support svn repositories as well. Because the repos reside in an external server, there is a wide range of client software for managing these repositories on all platforms.

Access to codebundles is controlled by a permission system. In contrast to Graphyte Atlas where all code was public within the system, code can be made private or shared with specific users or groups of users. The Graphyte system maintains separate permissions for reading, editing, and building codebundles to provide users fine-grained control of their collaboration.

7. DATA ACCESS AND THE BYTE BLOB SYSTEM

Data Access and storage within Graphyte is managed by the Byte Blob Data System (BBDS). Data inputs, artifacts from builds, results from runs, and libraries can all be interpreted as binary sequences of bytes or "byte blobs." These byte blobs can be stored on disk or in the cloud. The BBDS stores a list of redundant locations where these sequences of bytes can be retrieved, and these locations are indexed by a an automatically generated Byte Blob Identifier (BBID). The BBID is essentially a 256-byte random string made from two independent MD5 hashes. This is similar to a UUID but twice as long. Like UUID the sequence of bytes is so long that the probability of a collision between randomly chosen BBIDs is essentially zero.

The BBDS uses a non-relational document-oriented database, the open source MongoDB, to store associative arrays of metadata which refer to the BBID. This makes it possible to store arbitrary metadata about the byte blob. When a build or a run produces outputs, the artifacts are tracked in Graphyte by their BBID. The actual byte blob that is associated to that BBID may be stored externally, on disk, in the cloud, or distributed in many locations.

The BBDS is also used to allow users to access data from their programs. First the data that is to be used must be tagged with a BBID. For example, the user could upload the data in to the system. If the data is an artifact resulting from a run or build within the system, it already has a BBID. In the currently developed version of the system, libraries have been developed which allow users to access and add to a large collection of remote sensing data including MODIS and GOES imager data. Libraries are provided for reading standard data formats from NOAA and NASA such as NetCDF and HDF.

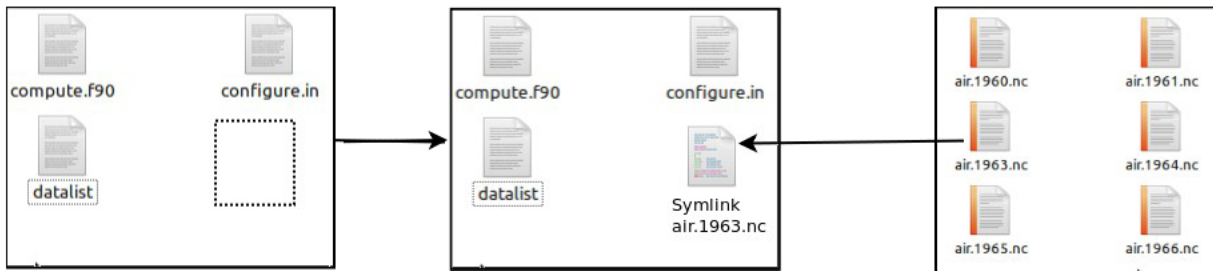


Figure 11. Diagram illustrating linking of data. The program assumes that, when run, a data file "air.1963.nc" appears in the same directory as the source (far left). Assuming the data is available on the computational server (far right), it will be linked so as to appear in the right location just before execution (center.)

Once data has been given a BBID it can be accessed by user code. Figure 11 illustrates how data in Graphyte is made available. A configuration file is generated when the user issues a command that the build (or run) should attach the data file with a particular BBID, and make it appear in the code directory with the name "air.1963.nc". As part of the process of preparing a codebundle for build or run on the computational server, the BBID is resolved to a location. In the far left of figure 11, if the BBDS system can locate a copy of the data file on the computational server, and that server supports symbolic links, then a link is created so that the data appears in the appropriate location to the program. This strategy is particularly useful for large data sets.

If data required by the codebundle is not available at the computational server before a run or build, it must be copied there. This is clearly not a problem for small data sets. For larger data sets, the data may be copied but be stored in a cache so that frequently used data need not be copied repeatedly. As a result, the first time a codebundle is built or run using a particular data set, the time to copy that data set will delay the execution. On subsequent builds or runs the data will be available, and only linking need occur.

8. BUILDS, RUNS AND CONDOR

As illustrated in figure 7, a Codebundle can have data attached to it and be transformed into a build model. The build model is a container object tracking a set of build outputs which are registered in the BBDS. When the language used is a statically compiled language such as C/C++, Fortran or Java, the result of a build is an executable of some form. In the case of a dynamic programming language like Python, Ruby, or MATLAB, the build process may not produce any outputs. A build must indicate a file to run, which may have been indicated in the codebundle metadata. In most aspects runs are identical to builds. A run results in run outputs which may be viewed or used in other runs, just like build ouputs. The difference is that whereas a build model supports a run action, the run model is terminal. It does not directly support an action which results in the creation of other runs. Both runs and builds are executed on a computational server. To support High Performance Computing (HPC), Graphyte manages jobs through the Condor High Throughput Computing System.

Condor is a system for managing intensive jobs across distributed computing resources, such as clusters, grids, and the cloud. It provides tools for scheduling jobs, matching jobs with a suitable system, monitoring the status of jobs, and taking advantage of idle resources without interfering with their normal use. A Condor cluster contains one machine that is the central manager, and many machines that submit and run jobs. Jobs are enqueued with a "ClassAd," a file describing the job and its requirements such as operating system, libraries or memory needed. The manager goes through the queue, sending jobs to suitable run machines. Once the job has finished, all the outputs are transferred back to the submitting machine. Condor provides a hooks system, so that on completion of jobs code can be executed. Using this hook system, Condor can communicate back to Graphyte the status or completion of jobs. Upon completion of a run or build, the BBDS tags and stores outputs and finalizes build and run models.

Condor can be used to send jobs to local clusters of machines, to grids, or to cloud computing systems. It can dispatch to all three depending on criteria. Thus, using Graphyte and Condor, users can build and run experiments which leverage HPC resources without needing to know the complexities of how to submit jobs or collect the outputs.

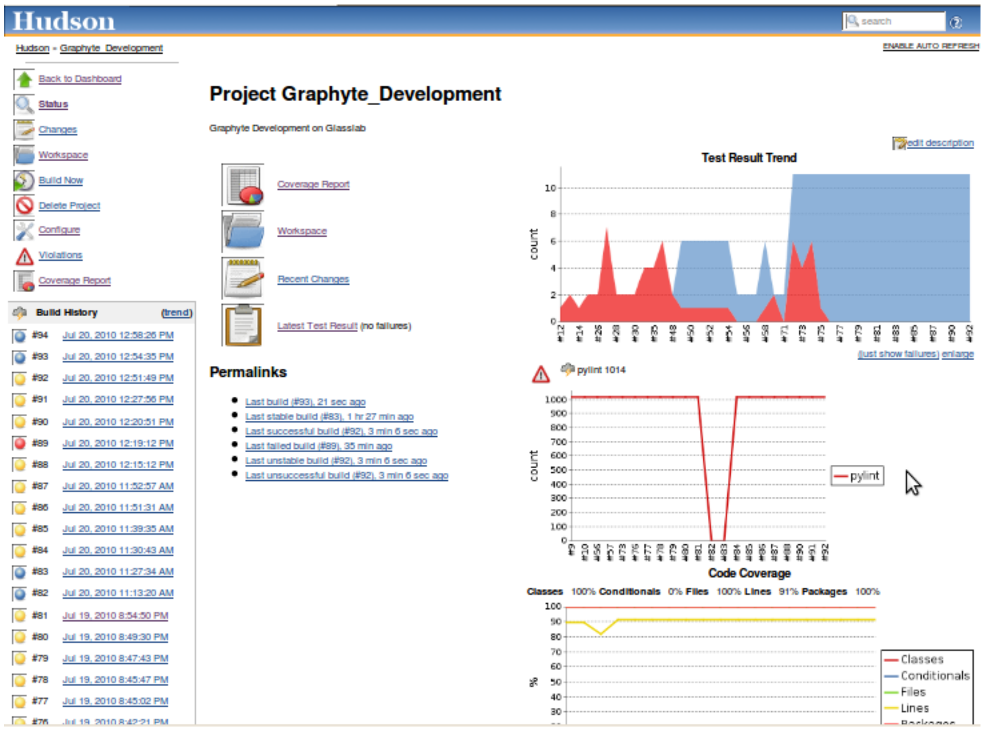
9. SOFTWARE ENGINEERING

Graphyte is being developed using a number of important software engineering "best practices." Since Graphyte is an open source project, as its user and developer community grows, it is important that it be managed in a sustainable way. In particular many of the tools and practices of agile software development are in use. One important factor is that Graphyte is primarily being developed in Python. Python is an open source and freely available programming language supported throughout government and industry. It binds well to other languages, making it possible to use the best libraries in a nearly language-agnostic way.

Graphyte server technology is being developed in Linux to be easily deployed in Ubuntu. The GCLI currently works in *nix environments such as Linux and MacOS X, but will be ported to work on Windows as well. The indirect client web application (server) runs in a Linux environment but is accessible from a modern web browser running on any operating system.

Graphyte source code is maintained using Mercurial for version control. In addition, unit tests are developed using the Python Nose library and the built-in unittest framework. Tests for the web components were developed using the Python Twill package, which allows for scripting interaction with a web interface. The Python coverage package is used to measure how much of the code is currently being tested. In addition the code is automatically run through static type checking to maintain consistent coding style. The project also uses automated deployment and sandboxed virtual environments to maintain consistency for testing. The project is built and tested automatically using the continuous integration server Hudson. Hudson pulls the project code from the Mercurial repository, builds the project through a script, runs all the tests, and generates a report. A screen shot of one report during development is shown in figure 12. The practice of automated builds and testing not only maintains quality but helps track progress and eases coordination between multiple developers.

While these practices cover the Graphyte software itself, rather than scientific experiments hosted through Graphyte, Graphyte does make it easier to institute these practices. For example, in order to use the system, user code must be checked into a version control system of some kind. In addition, many of the test frameworks and packages are available on the computational servers, removing the effort to install this software as a barrier to acceptance.



Project Graphyte_Development

Graphyte Development on Glasslab

Latest Test Result (no failures)

Build History (trend)

#94	Jul 20, 2010 12:58:26 PM
#93	Jul 20, 2010 12:54:35 PM
#92	Jul 20, 2010 12:51:49 PM
#91	Jul 20, 2010 12:27:56 PM
#90	Jul 20, 2010 12:20:51 PM
#89	Jul 20, 2010 12:19:12 PM
#88	Jul 20, 2010 12:15:12 PM
#87	Jul 20, 2010 11:52:57 AM
#86	Jul 20, 2010 11:51:31 AM
#85	Jul 20, 2010 11:39:35 AM
#84	Jul 20, 2010 11:30:43 AM
#83	Jul 20, 2010 11:27:34 AM
#82	Jul 20, 2010 11:13:20 AM
#81	Jul 19, 2010 8:54:50 PM
#80	Jul 19, 2010 8:49:30 PM
#79	Jul 19, 2010 8:47:43 PM
#78	Jul 19, 2010 8:45:47 PM
#77	Jul 19, 2010 8:45:02 PM
#76	Jul 19, 2010 8:42:21 PM

Permalinks

- Last build (#93), 21 sec ago
- Last stable build (#83), 1 hr 27 min ago
- Last successful build (#92), 3 min 6 sec ago
- Last failed build (#89), 35 min ago
- Last unstable build (#92), 3 min 6 sec ago
- Last unsuccessful build (#92), 3 min 6 sec ago

Classes 100% Conditionals 0% Files 100% Lines 91% Packages 100%

Figure 12. A screen shot from the Hudson continuous integration server which automatically builds, tests and generates a project status report.

10. CONCLUSION

We have refined our first version of Graphyte, “Atlas,” to bring together more libraries and analysis tools for remote sensing, and experimented with utilizing HPC. Through our experience with NOAA scientists and collaborators, we have identified issues which needed to be addressed to better accommodate NOAA and NOAA-related research. We have developed a new system “Betelgeuse” which addresses these issues. The new system provides fine-grained permissions, accommodates many programming languages, and supports using HPC for computation and distributed data management. Most importantly it has been designed so that it can be integrated into other systems. This means that as GEOSS-V is developed, Graphyte can provide components and services to help integrate the analysis of remote sensing and atmospheric data.

ACKNOWLEDGMENTS

The authors would like to thank Alfred Powell and Ingrid Guch of NOAA/STAR for their support and advice. Special thanks also to Ken Pryor, NOAA/STAR. This work was partially funded by NOAA as part of the “Using Graphyte to Create a platform for NOAA collaborative Scientific Computing” project, through NOAA/NESDIS CICS prime award NA09NES4400006, sub-award identifier MGMGHPCC-11 and by NOAA-CREST award NA06OAR4810162.

REFERENCES

[1] Grossberg, M., Gladkova, I., Guch, I., Alabi, P., Shahriar, F., Bonev, G., and Aizenman, H., "Rich client data exploration and research prototyping for NOAA," *Atmospheric and Environmental Remote Sensing Data Processing and Utilization V: Readiness for GEOSS III* **7456**(1), 74560C, SPIE (2009).

[2] David Stern and ITT Visual Information Solutions, "IDL: Interactive Data Language," (2010).

[3] University of Wisconsin-Madison Space Science UNIDATA and Engineering Center, "UNIDATA Man Computer Interactive Data Access System (McIDAS)," (May 2007).

[4] Callahan, S. P., Freire, J., Santos, E., Scheidegger, C. E., Silva, C. T., and Vo, H. T., "Vistrails: visualization meets data management," in [*Proceedings of the 2006 ACM SIGMOD international conference on Management of data*], *SIGMOD '06*, 745–747, ACM, New York, NY, USA (2006).

[5] Thomas Oinn and Mark Greenwood and et al Addis, "Taverna: lessons in creating a workflow environment for the life sciences," *Concurrency and Computation: Practice and Experience* **18**, 1067–1100 (August 2006).

[6] Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludäscher, B., and Mock, S., "Kepler: An extensible system for design and execution of scientific workflows," in [*Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*], 423–424, (2004).

[7] De Roure, D., Goble, C., and Stevens, R., "Designing the myexperiment virtual research environment for the social sharing of workflows," in [*e-Science 2007: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*], *Future Generation Computer Systems* , 603–610, IEEE Computer Society (10-13 December 2007).

[8] Klimeck, G., McLennan, M., Brophy, S. P., III, G. B. A., and Lundstrom, M. S., "nanohub.org: Advancing education and research in nanotechnology," *Computing in Science & Engineering* **10**(5), 17–23 (2008).

[9] De Roure, D., Goble, C., Bhagat, J., Cruickshank, D., Goderis, A., Michaelides, D., and Newman, D., "myExperiment: Defining the Social Virtual Research Environment," *eScience, IEEE International Conference on* **0**, 182–189, IEEE Computer Society, Los Alamitos, CA, USA (7–12 December 2008).

[10] Borner, K., "Plug-and-play macroscopes," *Commun. ACM* **54**, 60–69 (March 2011).

[11] Stein, W. et al., *Sage Mathematics Software.* The Sage Development Team (2010).

[12] Yasar, D., Yasar, S., and et. al (2008). www.kodigen.com, Accessed July 7, 2011.

[13] Pryor Kenneth. L., "Microburst nowcasting applications of GOES," *ArXiv e-prints* (jun 2011).

[14] Kanamitsu, M., Kistler, R., and et.al E. Kalnay, W. C., "The NCEP/NCAR 40-Year Reanalysis Project," *Bulletin of the American Meteorological Society* **77**, 437–477 (march 1996).