Automatic Provenance Collection and Publishing in a Science Data Production Environment—Early Results

James Frew, Greg Janée, and Peter Slaughter

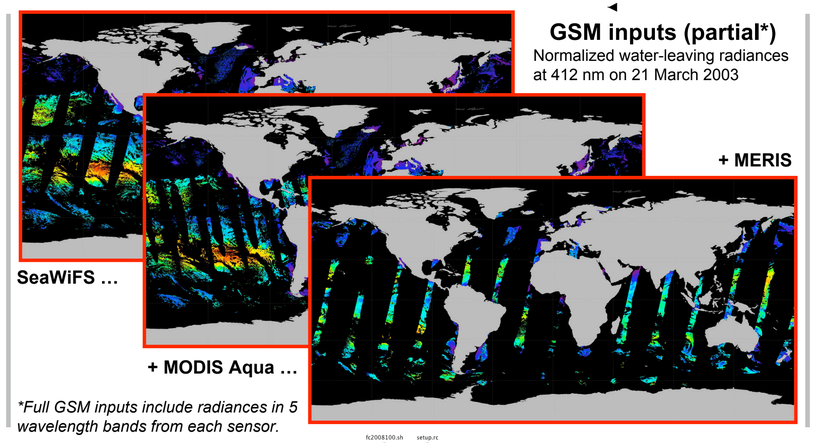
Institute for Computational Earth System Science, University of California, Santa Barbara

Introduction

Provenance is metadata about an object's origin and history. For an Earth science data product, such as a satellite-derived map of global ocean color, provenance comprises the source data sets, processing steps, and parameter settings that produced the final product. Provenance is conventionally recorded either manually (e.g., by a human taking notes), which is tedious and error-prone, or prescriptively, as in a workflow graph which must be then be executed in a specific workflow system. We have developed a third, passive approach to provenance capture as part of our Earth System Science Server (ES3) project (Frew *et al.*, 2007, DOI:10.1002/cpe.1247). Our passive provenance capture implementation works in any Linux-derived computing environment, without any changes to the science processing codes.

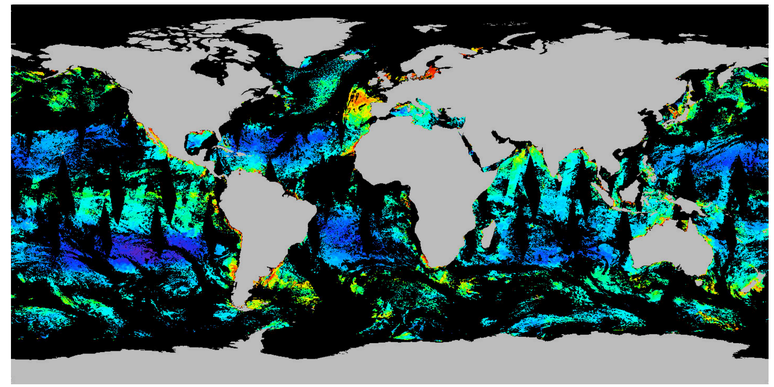
GSM ocean color model

The Garver-Siegel-Maritorena (GSM) model (Maritorena and Siegel, 2005, DOI:10.1016/j.rse.2004.08.014) uses normalized water-leaving radiances from any available sensors to invert a semianalytical ocean color model and simultaneously retrieve chlorophyll *a* concentration, combined dissolved/detrital particulate absorption (CDM), and particulate backscatter (BBP). GSM is unique in that it calculates the best combination of input sensors independently for each output value. The model is not computationally intensive, so its application to generating climate data records allows for frequent reprocessing. ES3 helps maintain the provenance of these multiple versions, without any changes to the current GSM implementation (a combination of UNIX shell scripts and interpreted IDL programs.)



GSM inputs (partial*)
Normalized water-leaving radiances at 412 nm on 21 March 2003

**Full GSM inputs include radiances in 5 wavelength bands from each sensor.*

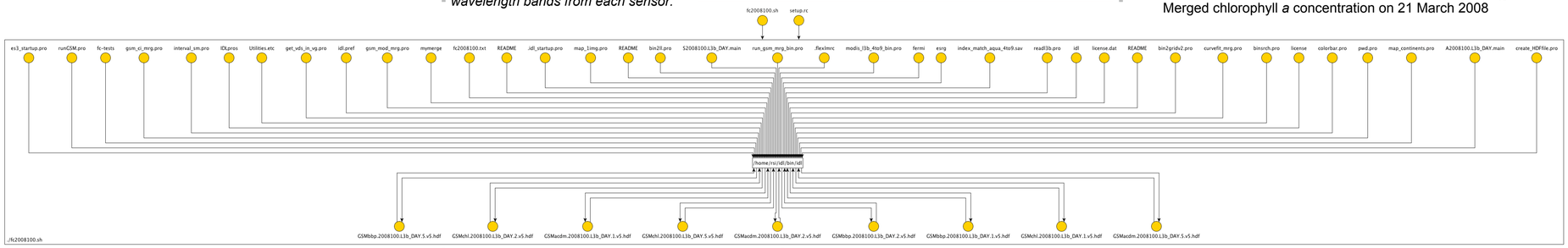


GSM output (partial*)
Merged chlorophyll *a* concentration on 21 March 2008

*(*GSM outputs also include CDM, BBP, and estimated error fields)*

What's this? →

The large "wiring diagram" across the center of the poster illustrates the complete processing history of a single run of GSM, as recorded and then reconstituted by ES3.



← Legend

- *Circles*: files read and/or written
- *Boxes*: programs executed.

Nesting indicates programs that execute other programs and wait for them to terminate (e.g., command interpreters)

- *Arrows*: data transfer (I/O)

(graph rendered by yEd)

ES3 architecture

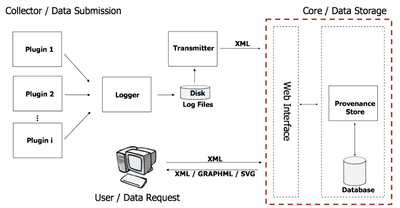
ES3's provenance capture system uses a client-server architecture.

The **collector** client augments a science process's execution environment (e.g., the *bash* shell) with **plugins** (e.g., *strace*) that monitor the science process's system calls.

The collector plugins communicate synchronously with a **logger** process that identifies provenance events (file access, program execution, etc.) and writes them to local log files.

When a logger process exits, a **transmitter** process scans the log files, assigns a universally unique identifier (UUID) to each provenance event, and formats each event (and any available metadata for the objects involved) as an XML message.

Provenance messages are submitted to the **ES3 core**: a provenance database with a web service interface.



Events in ES3

The request-response excerpt at right (end tags omitted for brevity) shows how an application can request information from ES3 about single events (in this case, those involving files that match the wildcard expression "*.hdf".) The response (single file excerpt) shows how, in addition to the usual filesystem metadata, ES3 saves the file's UUID, digital signature, and the UUID of the **workflow** (e.g., shell script) in whose context the file was accessed.

```
<ES3Request type="lookupFile">
  <select>
    <file>
      <where>
        <localId>
          <regex>
            .*hdf

<ES3response type="lookupFile">
  <file>
    <uuid>
      fd63a471-f686-467c-8dac-975cbc138036
    <size units="kb">
      0.0
    <owner>
      es3
    <localId>
      …/GSMchl.2008100.L3b_DAY.1.v5.hdf
    <timeOfLastAccess>
      20081122T061106Z
    <md5sum>
      e24d09744e4f26adb549035f520e7609
    <workflowUuid>
      2b9b441e-106e-45e1-8ab4-c7e792510fbb
```

```
<ES3Request type="getLineage">
  <traversal>
    <uuidStart>
      fd63a471-f686-467c-8dac-975cbc138036
    <direction> both
    <granularity> link
  <output>
    <format> graphml
    <formatOptions> nested,yfiles
    <detailLevel> full

<graphml …>
  <node
  id="fd63a471-f686-467c-8dac-975cbc138036"
  name="GSMchl.2008100.L3b_DAY.1.v5.hdf"
  objectType="file">
  …
  <edge
  source=
    "fd63a471f686-467c-8dac-975cbc138036"
  target=
    "55cd8e66-71a7-4a10-ac3b-cbb42e93cc14">
  …
```

Provenance in ES3

ES3 maintains provenance as a directed graph of input/output relationships between events. Provenance information is retrieved by specifying an event's UUID, a direction in which to proceed (forward, backward, or both), and an output graph representation. In the example to the left, the excerpted GraphML shows how objects and links in ES3 are represented as nodes and edges in GraphML.

Objects in ES3

ES3 manages provenance events, but those events usually involve objects (e.g., data or program files) that persist across events. ES3 therefore allows objects to be registered by an external name (currently, a filename.) This registration persists until the object's content is determined to have changed (e.g. in a "write" event.)

Identity Management for ES3

ES3's current use of filenames as external identifiers is expedient but problematic. Filenames are easily obtained by ES3, but there is no guarantee that a file won't be modified without ES3 being notified. Further, obtaining provenance for a file is currently a two-step process: obtain the file's UUID, then obtain the UUID's provenance.

For more robust external identity management, we are developing an extension for ES3 based on **Archival Resource Keys** (ARKs) (Kunze *et al.*, https://confluence.ucop.edu/display/Curation/ARK) ARKs allow a persistent, protocol-independent nomenclature that easily interoperates with both filenames and URLs. Object properties that would be published as *versions* will be embedded in ARKs. individual granule locators will be appended to these ARKs.

This work was supported by NASA Cooperative Agreement NNG04GE66G