# Provenance-enabled Automatic Data Publication

**James Frew, Greg Janée, and Peter Slaughter**
*Earth Research Institute, University of California, Santa Barbara*

## *ES3 Background*

**ES3** is a software system for automatically and transparently capturing, managing, and reconstructing the **provenance** of arbitrary, unmodified computational sequences. **Automatic** acquisition avoids the inaccuracies and incompleteness of human-specified provenance. **Transparent** acquisition avoids the computational scientist having to learn, and be constrained by, a specific language or schema in which their problem must be expressed or structured in order for provenance to be captured.

Unlike most other provenance management systems, ES3 captures provenance from running processes, as opposed to extracting or inferring it from static specifications such as scripts or workflows. ES3 provenance management can thus be added to any existing scientific computations without modifying or re-specifying them. ES3 models provenance in terms of **processes** and their input and output **files**. A process is a specific execution of a program. In other words, each execution of a program or workflow, or access to a file, yields new provenance events. Relationships between files and processes are observed by monitoring read and write system calls (using `strace`). This allows ES3 to function completely independently of the scientist's choice of programming tools or execution environments.

An ES3 provenance document is the directed graph of files and processes resulting from a specific **job** (invocation event). Nested processes (processes that spawn other processes) are correctly represented. In addition to retrieving the entire provenance of a job, ES3 supports arbitrary forward (descendant) and/or reverse (ancestor) provenance retrieval, starting at any specified file or process.

## *Data publication*

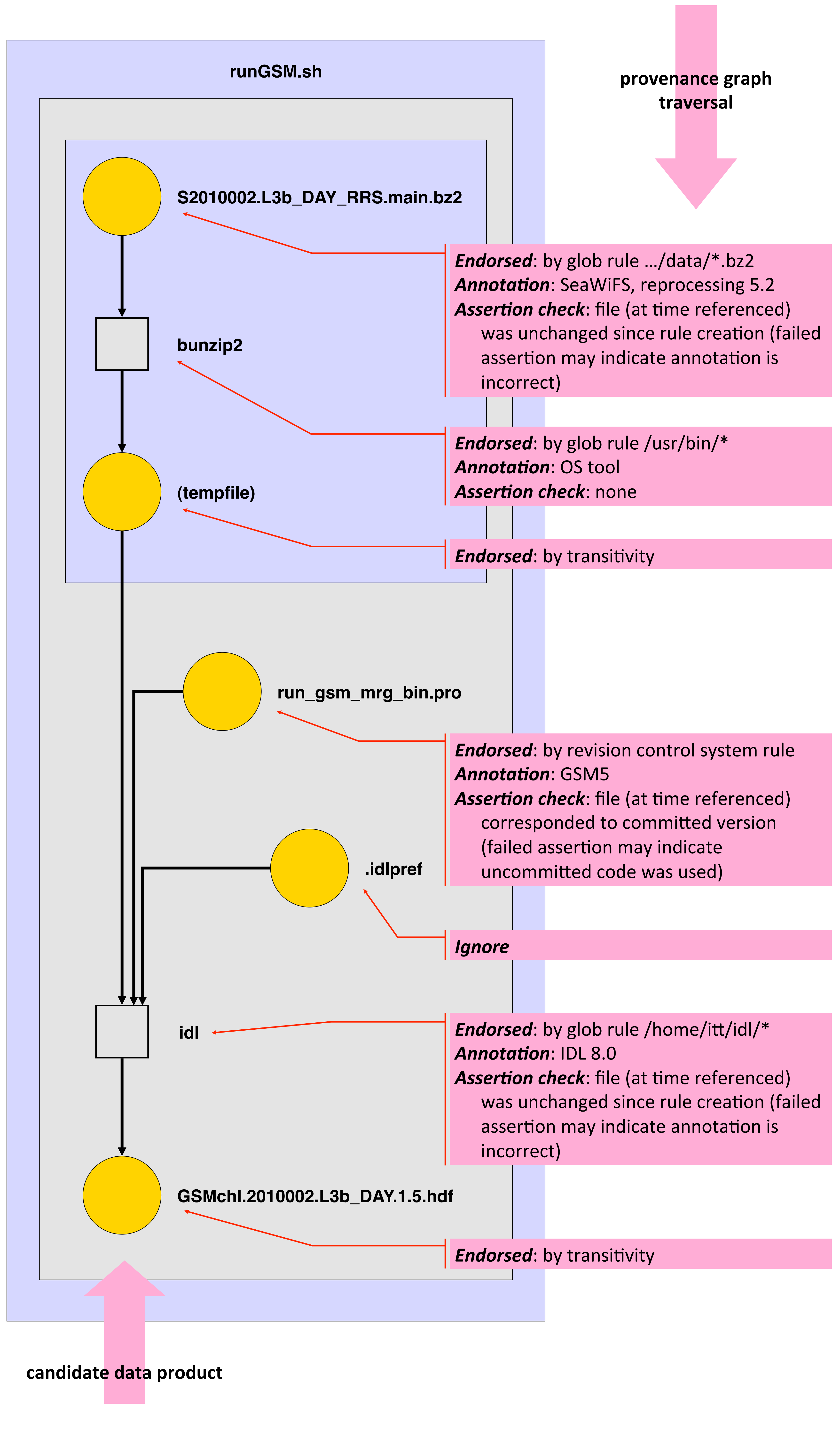
We assume that an object's fitness for publication is based largely on the process by which that object was produced, and the components which were combined or transformed in order to produce it. An object produced by scientific codes with known significant bugs, or from source datasets with known errors, may not be suitable for publication, regardless of how well formatted, documented, and presented it is. Or, even if the codes and data used to produce an object are acceptable, they may not be suitably curated, and this lack of a guaranteed level of future availability to users of the object may disqualify it for publication.

**Provenance-driven publication** thus involves traversing a candidate object's provenance to some suitable depth and evaluating **assertions** as to whether the object's antecedents justify a decision to publish the object. The assertions are configurable and will depend greatly on the context, but generally fall into two categories:

**Provenance**, or, What did we do? Do we have sufficient information describing the processing that would allow a reader to unambiguously re-create the processing, in principle if not in actuality?

**Confirmation**, or, Did we really do what we think we did? A publishing system should catch such mistakes as operating on the "wrong" file, because the file has the same name as the correct file or because the file's contents have changed without our knowledge.

The publish tool can also automatically annotate the provenance graph, thus bridging the gap between the level at which ES3 operates (files) and higher-level concepts that may be more meaningful to users (datasets, software packages).


runGSM.sh
provenance graph traversal
S2010002.L3b_DAY_RRS.main.bz2
bunzip2
(tempfile)
run_gsm_mrg_bin.pro
.idlpref
idl
GSMchl.2010002.L3b_DAY.1.5.hdf
candidate data product

***Endorsed***: by glob rule .../data/*.bz2
***Annotation***: SeaWiFS, reprocessing 5.2
***Assertion check***: file (at time referenced) was unchanged since rule creation (failed assertion may indicate annotation is incorrect)

***Endorsed***: by glob rule /usr/bin/*
***Annotation***: OS tool
***Assertion check***: none

***Endorsed***: by transitivity

***Endorsed***: by revision control system rule
***Annotation***: GSM5
***Assertion check***: file (at time referenced) corresponded to committed version (failed assertion may indicate uncommitted code was used)

***Ignore***

***Endorsed***: by glob rule /home/itt/idl/*
***Annotation***: IDL 8.0
***Assertion check***: file (at time referenced) was unchanged since rule creation (failed assertion may indicate annotation is incorrect)

***Endorsed***: by transitivity

## *Approach and operation*

The publish tool **endorses** a data product proposed for publication, by affirming that the data product's provenance has been examined and is of sufficient integrity that the product is worthy of publication. Endorsement is **transitive** in the sense that a node (file or transformation) in the provenance graph is fully endorsed only if the node itself is endorsed and all antecedents of the node are endorsed. A node that is itself endorsed but has one or more unendorsed antecedents is considered "partially" or "provisionally" endorsed. A **comment** may optionally be associated with the endorsement. Both quantities are stored in the provenance graph for future reference.

The publish tool performs a depth-first, post-order traversal of the provenance graph starting from the data product's node. At each node, the user is presented with the following options:

**Endorse**. A comment may optionally be added describing the file or transformation, the node's provenance, etc.

**Ignore**. Affirm that the node is insignificant to the data product's provenance.

**Skip**. Postpone making a decision about the node and continue traversing the provenance graph. All subsequent endorsements will be considered provisional until the skipped node is addressed.

With the above options, endorsement is entirely manual. Three types of rules help to automate the process:

**Filename patterns**. The user can specify that all objects whose filenames match particular pattern (a **glob rule**) should be automatically endorsed. For example, a glob rule might state that all files in a certain directory are endorsed. Glob rules keep track of the files matching the pattern so that the user can be alerted if a file covered by a rule has been added or changed since the rule was created.

**Source code repositories**. Files in revision control systems (currently, Mercurial) are recognized as. If a file corresponds to a committed version in a repository, endorsement is automatic, with the repository and file's version as the comment; otherwise, the user is warned that uncommitted source code was used.

**Transitivity**. If all inputs to a transformation are endorsed, and if the transformation itself is endorsed, then the outputs of the transformation are automatically endorsed.

With these rules in place, endorsement becomes largely automatic.

## *Issues and next steps*

What is the "right" **granularity** for provenance? ES3 records provenance the level of file operations. A transformation may open and close the same file multiple times within a single execution context, sometimes for reading and sometimes for writing. Such access patterns cause cycles in the provenance graph: the file is an antecedent of the transformation and the transformation is an antecedent of the file. At a coarser granularity we could collapse cycles by abstracting multiple file accesses into a single relationship between a transformation and a file. For example, if a transformation initially creates a file, we can safely summarize that the transformation produced the file. The issue here is finding a granularity commensurate with the publish tool's needs.

How can relationships between **source files and their compiled artifacts** be maintained? We want to track which versions of source files were used by transformations, but in compiled languages (including interpreted languages like Python that compile source code on-the-fly) transformations access compiled artifacts, not source files. We may thus require publication rules that explicitly encode certain file types and their relationships.

What is the version of a file in a **distributed revision control** system? Mercurial and Git store revisions as "changesets," each of which may contain multiple files, and which are linked to form an acyclic graph. A particular revision of a file may be explicitly contained in multiple changesets and implicitly inherited by many others. The publish tool asks: what revision of a file was used at some execution time in the past? Strictly speaking, any and every changeset that includes the revision of the file, directly or indirectly, is a possible answer, but our goal is to find the one best version for the file. We believe the answer may be determined by a combination of analyzing the revision graph, finding the earliest changeset in the graph at which the file revision appears, and then examining timestamps and execution times.

*This work was supported by NASA Cooperative Agreements NNG04GC52A and NNX08AP36A*