DataONE RunManager: Capturing Provenance of R and Matlab Scripts

Alfred Hofmann*, Ursula Barth, Ingrid Haas, Frank Holzwarth,
Anna Kramer, Leonie Kunz, Christine Reiß,
Nicole Sator, Erika Siebert-Cole, and Peter Straßer

Springer-Verlag, Computer Science Editorial,
Tiergartenstr. 17, 69121 Heidelberg, Germany
`{alfred.hofmann,ursula.barth,ingrid.haas,frank.holzwarth,`
`anna.kramer,leonie.kunz,christine.reiss,nicole.sator,`
`erika.siebert-cole,peter.strasser,lncs}@springer.com`
`http://www.springer.com/lncs`

Abstract. The abstract should summarize the contents of the paper and should contain at least 70 and at most 150 words. It should be written using the *abstract* environment.

Keywords: We would like to encourage you to list your keywords within the abstract section

1 Introduction

Scientific workflow provenance tracking is well recognized as a valuable approach in computational science. The term *provenance* originally is used in the art world to establish the authentication of a work of art. Provenance has been used in other fields as well. For example, provenance in the field of the scientific workflow refers to having knowledge of all the steps involved in producing a scientific result. Such information can help scientists to better understand their work. Additionally, scientists can document their experiments and share experimental data and computational steps with other researchers. These activities can be called *reproducible research.*

Scientific workflow systems support two aspects of provenance information: *prospective* and *retrospective* provenance. Prospective provenance presents a workflow as a directed graph with data and planned computational steps, while retrospective provenance captures the context of a script execution.

There are many workflow management systems such as Taverna, Kepler, Galaxy that can transparently track, query, and visualize the scientific provenance information such as intermediate results, workflow steps, and parameter settings. But many scientists prefer to use programming languages (such as R,

* Please note that the LNCS Editorial assumes that all authors have used the western naming convention, with given names preceding surnames. This determines the structure of the names in the running heads and the author index.

Matlab, Python) for scientific computing work without using workflow management systems. For this end, several provenance tracking systems (such as Sumatra [7] [11], noWorkflow [1] and RDataTracker [8]) have been implemented to assist scientists to capture and analyze the provenance traces of script execution.

The existing work for capturing provenance for scientific workflow is presented in Section 2. We have developed a DataONE *RunManager* for capturing the prospective and the retrospective provenance traces of R and Matlab script executions. In this paper, the design and implementation details of the DataONE RunManager is described in Section 3. The demo for the features of the RunManager is explained in Section 4. Future work and conclusion is summarized in Section 5.

2 Related Work

There have been many efforts to develop automated provenance systems to capture different levels of provenance information as scripts run. Some researchers classify the existing provenance capture systems into three categories: literate programming, workflow management system, and environment capture systems [11]. Existing environment capture systems are discussed in this section.

1. Use virtual machine to record everything

VM

Advantages:

Limitations: the file size is too large. Or there is no way to search, index, and query provenance information in this approach

2. Code, data, environment

CDE [10]

Advantages:

Limitations:

The Sumatra [7] system is a Python toolkit for scientific workflow reproducible research. It uses version control system to capture the information about all of the code that was run, such as the local file system path of each library/module/package that is included/imported by the "main" file, together with the version of the modules [11]. For input data, Sumatra can track data either reading from standard input or requiring users to annotate the location of input data. For output data, users need to instruct Sumatra the path of a directory to monitor. Metadata associated with provenance record with metadata are required for searching. Command-line interface and built-in local web server interface are provided for search/query/resue of the provenance records.

The noWorkflow [1] system is a Python tool which transparently captures provenance of scripts. It relies on the techniques of abstract syntax tree analysis, reflection, and python profiling to collect fine-grained provenance. Rich account of the data dependencies is helpful. But the amount of information can be overwhelming and it is difficult to extract and organize information. Additionally, it does not rely on calls to provenance capture functions from users. Also, the metadata feature is absent for the noWorkflow system.

The RDataTracker [8] system is an R library that supports the collection of data provenance generated by the execution of R scripts and R commands entered interactively at the R console. RDataTracker requires users to adds calls to functions defined in the RDataTracker library to collect data provenance as an R script executes. The provenance data is stored in a Data Derivation Graph (DDG).

3 Approach

The DataONE RunManager has been implemented in the NCEAS *recordr* R package [4] and the *Matlab DataONE Toolbox* [5] to enable systematic capture of a script execution and console commands without the need to modify existing R/Matlab code. The provenance captured during a script execution includes information about the script that was run, files that were read or written, and details about the execution environment at the time of execution. A package of the script itself, its input files, and generated output files associated with the run can be retrieved for a given run in order to investigate previous versions of processing or analysis and support reproducibility. The execution's packaged artifacts can be easily published to a repository within the DataONE network. The DataONE RunManager also provides a command-line interface for viewing the provenance records in text format.

3.1 DataONE RunManager

The DataONE RunManager is used to track both prospective and retrospective provenance in order to provide a complete understanding of a scientific script in terms of script structure and script execution. RunManager implementations determine provenance relationships between objects related to a script execution and record those relationships in a standard format. Captured provenance information is archived in the local file system and can be published to the DataONE network for data sharing.

The principal classes in the matlab-dataone toolbox and their composition are shown in Fig. 1. The labels on the arrows denote functions that can be used between different modules in the RunManager. Firstly, an Execution represents a script run, and contains some critical metadata needed to understand the execution environment, uniquely identify the run, categorize it, and know it's start and end times. Secondly, the Configuration class allows the scientist to configure the RunManager with properties such as the provenance information storage directory, DataONE target member node, DataONE coordinating node, and etc. The Configuration class contains an instance of ScienceMetadata and ywConfiguration. Thirdly, the DataONE DataPackage is used to store an RDF graph of the execution's members, and is serialized in the Open Archive Initiative's Object Re-use and Exchange (ORE) [3] format. Input and output file objects are

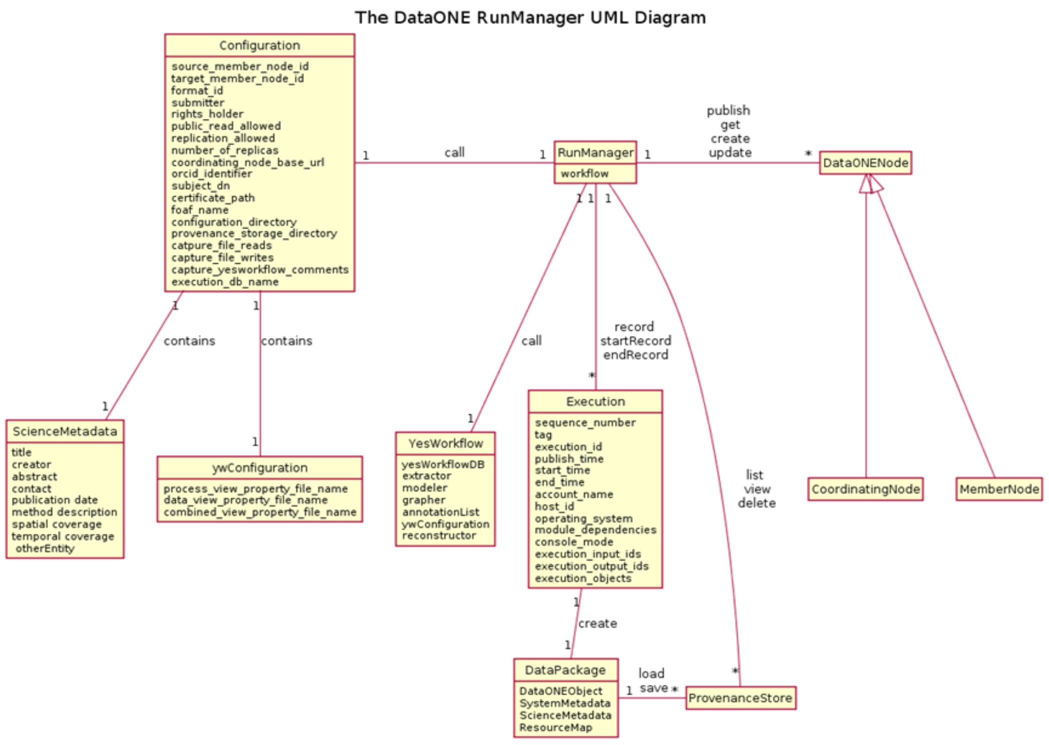

Fig. 1. The principle classes with their attributes in matlab-dataone toolbox which is an implementation of the DataONE RunManager using Matlab and Java. In order to save space, the attributes for the *CoordinatingNode*, *MemberNode*, *DataONEObject*, *SystemMetadata* and *ScienceMetadata* classes are not shown.

archived as DataONEObjects with system-level metadata that describe the objects. The RunManager assists in the preparation of science metadata, currently in the Ecological Metadata Language (EML) format, that describe the input and output science data objects associated with a script execution. Therefore, an DataPackage instance contains a list of instances of DataONEObjects, SystemMetadata, and ScienceMetadata. Lastly, the DataONENode class represents a DataONE Node and it is a superclass to the MemberNode and CoordinatingNode classes, providing shared DataONE API functions that can make provenance data available in the DataONE network.

There are two ways to store the provenance data in the DataONE RunManager. At the current stage, data is stored in individual files on the local file systems for the matlab-dataone toolbox. But data is stored in a relational database for the recordr R package: there is only one copy for the input/output files. The copy of a file object is deleted when there is no reference pointing to it.

In the DataONE RunManager, prospective provenance is captured by the YesWorkflow (YW) [2] Java library which models conventional scripts and programs as scientific workflows. Instead of reimplementing code to be executed and managed by a workflow management system, a user embeds YW annotations within an existing script. These annotations declare how data will be used and derived by the script step by step. The YW library interprets the annotations and builds a graphical output that represents the script as a collection of entities, i.e., programs, workflows, ports, and channels which are mapped to the DataONE ProvONE model shown in [9].

DataONE has designed the ProvONE provenance data model to extend the W3C PROV data model [6] to express detail about the processes, ports, and data links common to scientific workflow systems. Both RunManager implementations in Matlab and R produce RDF provenance output adhering to the ProvONE data model so that the provenance information generated from different programming languages like R and Matlab can be shared in a common format.

The DataONE RunManager implementation automates the collection of retrospective provenance data. The automated collection is transparent to scientists and provides a medium-grained, rich account of the data dependencies. More fine-grained automation approaches used in noWorkflow and RDataTracker result in voluminous provenance data. In our work, we currently focus on collecting file-level input and output data.

3.2 Overloading I/O Functions

Our approach relies upon two basic techniques. First, we developed wrapper functions for commonly-used I/O functions in R and Matlab so that provenance information can be recorded as a script executes. The wrapper functions collect and store provenance information after calling the original I/O function. Second, during the initial installation of the R or Matlab libraries, the paths to these overloaded I/O functions are added to the function search path so that calls to original I/O functions are directed to the overloaded functions with the same function name in the library.

Algorithm 1: Steps for overloading a Matlab function in the Matlab DataONE Toolbox

```
   Input: A number of input arguments $varargin = \{in_1, in_2, \ldots, in_n\}$
   Output: A number of input arguments $varargout = \{out_1, out_2, \ldots, out_m\}$, two
           updated input/output Id lists, and an updated execution object map
1  Get the RunManager instance
2  Remove the path to the overloaded functions from Matlab path
3  Call the native function with the same name
4  Add the path to the overloaded functions back to Matlab path
5  if the current file is a new data object then
6      Add this object to the execution objects map
7  else
8      Update the existing map entry with a new DataObject
9  Add the identifier of the current file to either the execution_in_ids or
   execution_out_ids set depending on the operation type
10 return {$out_1, out_2, \ldots, out_m$, execution_input_ids, execution_output_ids,
   execution_objects}
```

A set of functions to read and write data to files of different formats types are supported in the Matlab Dataone Toolbox and the recordr R package. The supported file formats include: MAT-files, text files, XML documents, Excel spreadsheets, scientific data in CDF, FITS, and HDF formats, audio/video files, graphical files, internet files, and DataONE get(), create(), update() API calls.

The steps for overloading a Matlab function is shown in Algorithm 1. The method of overloading functions in R is different that in Matlab. When overloaded functions defined in the *recordr* R package are to be used, a user only need to call *library(recordr)* in the R console. The R RunManager implementation overloads functions that read input and write output in order to capture the objects that are used and generated by the script execution. For instance, when a script reads in a CSV file, both RunManager implementations in Matlab and R can infer the triple *scriptExecution* $\rightarrow$ *prov:used* $\rightarrow$ $.csvfile$. The following PROV [6] provenance relationships are supported: prov:wasGeneratedBy, prov:used, prov:wasDerivedFrom, and prov:wasInformedBy.

3.3 Rendered at Web UI

4 Demo

Use Cases

1. Scientist Alice documents her script
2. Alice records the provenance of a script execution

3. Alice lists the given provenance information

4. Alice views a specific provenance

5. Alice publishes a provenance package to DataONE and shares the data package with other researchers

6. Scientist Bob views Alice's data package in web UI

7. Bob navigates between provenance relationships and downloads a whole data package

8. Interoperability between scripts in various programming lanaguages (R and Matlab)

Because the resource map is in RDF format which is independent of platformat

5 Future Work and Conclusions

References

1. Murta L, Braganholo V, Chirigati F, Koop D, Freire J. noWorkflow: Capturing and analyzing provenance of scripts. In: Ludäscher, B., Plate, B. (eds.) IPAW 2014. LNCS, vol. 8628, pp. 71-83. Springer, Verlag. (2015)
2. McPhillips T., Song T., Kolisnik T., Aulenbach S., Belhajjame K., Bocinsky R.K., Cao Y., Cheney J., Chirigati F., Dey S., Freire J., Jones C., Hanken J., Kintigh K.W., Kohler T.A., Koop D., Macklin J. A., Missier P., Schildhauer M., Schwalm C., Wei Y., Bieda M., and Ludascher B. YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts. International Journal for Digital Curation, vol. 10, 298–313. (2015)
3. Open Archives Initiative Object Reuse and Exchange. `https://www.openarchives.org/ore/` (2008)
4. Slaughter P., Jones M., Jones C. NCEAS recordr: Provenance tracking for R. `https://github.com/NCEAS/recordr` (2016)
5. Jones C., Cao Y., Slaughter P., Jones M. Matlab DataONE Toolbox. `https://github.com/DataONEorg/matlab-dataone` (2016)
6. PROV-DM: The PROV Data Model. `http://www.w3.org/TR/prov-dm/` (2013)
7. Davison A. Automated Capture of Experiment Context for Easier Reproducibility in Computational Research. Computing in Science & Engineering, vol.14, no. 4, pp. 48–56. (2012)
8. Lerner B., Boose E. RDataTracker: Collecting Provenance in an Interactive Scripting Environment. In: 6th USENIX Workshop on the Theory and Practice of Provenance, USENIX Association, Cologne. (2014)
9. Walker L., Jones C., Jones M. The DataONE PROV Ontology and Model. `https://github.com/DataONEorg/sem-prov-design/blob/master/docs/PROV-annotation-model/PROVmodel.md` (2015)
10. Guo P. and Engler D. CDE: Using System Call Interposition to Automatically Create Portable Software Packages. In: Proc. 2011 Usenix Annual Tech. Conf., Usenix Assoc. (2011)
11. Davison A.P., Mattioni M., Samarkanov D. and Teleńczuk B. Sumatra: A Toolkit for Reproducible Research. In: Implementing Reproducible Research, edited by V. Stodden, F. Leisch and R.D. Peng, Chapman & Hall/CRC: Boca Raton, Florida., pp. 57-79. (2014)