1

Dummy chapter needed for the \chapterauthor command to work later

CONTENTS

2

SOLE: Towards Descriptive and Interactive Publications

Tanu Malik
Computation Institute
University of Chicago and Argonne National Laboratory

Quan Pham
Dept. of Computer Science
University of Chicago

Ian Foster
Computation Institute
Dept. of Computer Science
University of Chicago and Argonne National Laboratory

CONTENTS

Prior to the computational-driven revolution in science, research papers provided the primary mechanism for sharing novel methods and data. Papers described experiments involving small amount of data, derivations on that data, and associated methods and algorithms. Readers reproduced the results by repeating the physical experiment, performing hand calculation, and/or logical argument.

The scientific method in this decade has become decisively computational, involving large quantities of data, complex data manipulation tasks, and large, and often distributed, software stacks. The research paper, in its current text form, is only able to summarize the associated data and computation rather than reproduce it computationally. While papers

corroborate descriptions through indirect means, such as by building companion websites that share data and software packages, these external websites continue to remain disconnected from the content within the paper, making it difficult to verify claims and reproduce results. There is an critical need for systems that minimize this disconnect.

We describe Science Object Linking and Embedding (`SOLE`), a framework for creating descriptive and interactive publications by linking them with associated *science objects*, such as source codes, datasets, annotations, workflows, re-playable packages, and virtual machine images. `SOLE` provides a suite of tools that assist the author to create and host science objects that can then be linked with research papers for the purpose of assessment, repeatability, and verification of research. The framework also creates a linkable representation of the science object with the publication and manages a bibliography-like specification of science objects. In this chapter, we introduce `SOLE`, and describe its use for augmenting the content of computation-based scientific publications. We present examples from climate science, chemistry, biology, and computer science.

2.1 Introduction

Scientific publication has been central to scientific practice since the 17th Century. It provides a means to describe the methods used in a scientific study and to argue the scientific logic. Over the centuries, the form of publication has changed little. Yet with the emergence of the electronic computer, the methods used in science have changed dramatically, with almost all scientific projects now involving some amount—often large amounts—of computation, to run simulations and/or to extract, analyze, and store information. The static text-based publication form is seriously insufficient for the current scientific method. It is inadequate for helping the reader grasp descriptions, follow logic, or interpret results.

To cope with the limitations of the static form of publications, scientists adopt alternate mechanisms for augmenting the text-based publication. In particular, they include citation links to input and output data and software packages that may be shared through websites. However, despite this inclusion of links, the content in these Web resources remains largely disconnected from the claims made in papers—not allowing, for example, a parametric equation in a paper to be mapped directly to its implementation or its behavior studied under different parameters. This disconnect seriously affects readability of a computational publication and limits readers and reviewers from being able to assess the validity of findings, determine workability of results on different parameters, or reuse data and methods for their own research. It is an implied understanding in scientific domains that a true assessment of a computational publication is only feasible if readers have in-depth knowledge of the computational tools used in the research. It is only thus that mere descriptions are sufficient and an out-of-band communication channel with the authors becomes less important.

To improve readability of computer programs, Donald Knuth proposed literate programming [22], a macro-based mechanism that intermingles descriptions of abstract concepts with source code. Thus source code is read in a sequence that gives the best possible exposition of the code instead of the machine-imposed sequence as required by the programming language. Tools such as Web, CWeb, and NoWeb that adopt the literate programming approach produce compilable source code with one command (*tangle*) and description with another (*weave*). Such tools generate more meaningful documentation that can easily be maintained as software programs and libraries undergo revisions[65, 51].

Literate programming can also be used for assessing claims in a publication. A exemplar is the Sweave [26, 27] tool that allows an author to embed R code for complete data analyses

in Latex documents. Readers and reviewers can assess the validity of the figures and plots in a publication by executing the Sweave file in their local R environment. While Sweave serves as a verifiable framework for reproducing research described in papers, authors must adopt the Sweave framework prior to authoring papers. This requirement becomes an impediment for most authors, who often publish first and make publications reproducible post hoc, perhaps in response to their being highly cited by the wider research community. In addition, computation-based publications must increasingly make content associations beyond those enabled through literate programming in order to improve readability and assessment.

An approach toward making computation-based publications reproducible post hoc is to augment the content in the paper with computational and data elements of research. For instance, pseudo code can be augmented with an accompanying source code implementation; data values can be augmented with annotations; dataset descriptions can be augmented with metadata or even download history, thus making the publication more descriptive and readable. Similarly, publications that describe performance, robustness, or stability of approaches can be augmented with an accompanying package or environment to test the workability of the procedures under varying parameters. Such materials can help the reader to interact with and assess experiments. Finally, some publications reach conclusions on a scientific method being executed/simulated in a specialized environment such as a supercomputer or a cluster that cannot be replicated. Verification of such conclusions can only be made if a trustworthy provenance trace of the experiments is associated with the publication. Ideally, the reader should be able to replay the provenance trace.

We face at least three technical challenges when we seek to associate papers with computational artifacts in a descriptive or interactive manner. First, we need to transform each computational artifact into a form amenable to linkage with a paper. In our work, this means that important classes and functions in source code files must be associated with URLs and that datasets must be recorded in registries that specify dataset locations and access methods. It also means that data analysis pipelines must be cast as workflows with appropriate interfaces that specify inputs and functional forms, or alternatively, must be associated with software on an adequately provisioned virtual image. A second challenge concerns the manner in which linkages are represented in papers. Using URLs to refer to computational artifacts is often unwieldy, especially when an object is referenced multiple times. A third challenge relates to presentation: Clicking on a link for a computational artifact should lead to an adequate presentation to the user. These challenges are primarily technical in nature, but addressing them is vital for authors.

In this chapter, we describe Science Object Linking and Embedding (`SOLE`), a system that addresses these challenges and in doing so makes it convenient to link research papers with associated computational artifacts, such as source codes, datasets, workflows, repeatable packages and virtual images. `SOLE` provides tools that turn data, methods, and processes used in a scientific method to granular units of knowledge termed *science objects*. Authors identify science objects with human-readable tags; the tools convert each tagged science object into an associated linked data object identified by a URI. For ease of management, the tags, URIs, and accompanying representation are maintained in a metadata repository: what is, in effect, a science object bibliography. To aid authors with the linking process, `SOLE` also provides a web interface that allows authors to associate groups of words in a research paper with one or more science object tags. Clicking a link in the text results in the display of an appropriate representation of the science object. In addition, `SOLE` persists these objects, which can be accessed and executed on-demand by users to validate or repeat an experiment in a publication.

Through `SOLE`, we make the following contributions toward the emerging area of reproducible research:

1. We describe a comprehensive system based on tools that lets authors convert their computational products to a form amenable to linking with their publications. These command-line tools are easy to use and can be adopted by authors anytime—either while investigating the scientific method or post hoc after the publication is written.
2. The author need not change their authoring environment, but simply include packages or use an application that associates the `SOLE` system as a bibliographic tool of computational products.
3. We demonstrate the efficacy of this framework through various examples taken from the domains of climate science, biology, chemistry and computer science.

The remainder of the chapter is organized as follows: Section 2.2 describes related work. In Section 2.3 we describe the overall framework of `SOLE` and give an overview of the different science objects that can be used to augment a publication. We then describe methods and tools to create science objects and demonstrate how they can be used to augment publications (Section 2.4). In Section 2.5, we describe a specialized science object that can be used toward peer-review and repeatability testing. We present a discussion on reproducible research in 2.6, and finally conclude in Section 2.7.

2.2 Related Work

The first step towards reproducible research is to have open data and code. Thus initiatives such as Open Data [41], Open Source [42], and Open Access [40] thus all contribute to the meaningful association of computational products with publications. The Panton Principles lay out the principles for Open Data sharing, permitting any user to download, copy, analyze, reprocess software, or use data for any other purpose without financial, legal, or technical barriers [36]. Sparselab and Wavelab from Stanford toolkits demonstrate the impact of open code and open access on an entire field of research [10]. The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) and Open Archives Initiative Object Reuse and Exchange (OAI-ORE) translate some of these principles by exposing repository interfaces through structured metadata, thus making them compatible with reproducible research [25, 24]. eScience projects such as NanoHub [21] and RunMyCode.org [62, 52] (also described in the chapter by Stodden *et. al.*) allow sharing of code and applications, while open-access journals, such as PloS ONE [45], Giga Science [15], and BioMed Central [4] contribute to unrestricted access via the Internet to peer-reviewed scholarly journal articles.

Open is not sufficient; code and data must be linked with research papers. Several projects have demonstrated the concept of reproducible research papers by focussing on one or more aspects. Utopia [2] associates concepts in a paper with external annotations retrieved from an online meta data store. Annotations are publicly shared and readers can further comment upon them. The chapter on VisTrails [23] by Freire *et. al.* associates figures and results in the paper with executable components. It allows authors to publish workflows and associated provenance and hyperlink to these artifacts in the article. Workflow systems such as Taverna [39], MyExperiment [16], Kepler [1], Swift [69], and Galaxy [17] also capture provenance, but do not link provenance to publication as in VisTrails. However, as demonstrated by Gavish and Donoho, computational results from these systems can be referenced via a URL and thus archived and published as public repository services

for everyone to access [14]. Similarly, the Collage authoring environment allows a researcher to publish scientific experiments through scripts and usable forms [38].

Sweave [26, 27] and Dexy [9] are literate programming environments, which if adopted from the beginning of the scientific process can lead to papers with embedded source code and derived results. The Author-Review-Execute Environment shows how to achieve repeatability through a notion of linked results, but it locates the results archive on authors' own machines [33]. This approach requires that each author maintain a server and install the service to expose the data and execution resources, which raises issues of security and adoption. The SHARE environment creates a virtual image providing the execution services directly on its server [66].

An increasing number of conference and journal publications now mandate the need for repeatable experiments so that the reviewers and editors can re-run the exact same experiments with the same method on the same or similar system, and obtain the same or similar results. This approach has been deemed important for computer systems research in which measurement of real systems is important [29]. Conferences such as ACM SIGMOD have experimented with requiring repeatability [29, 28, 13]. However, as documented by recent experiments in repeatability testing, testing can be arduous and time consuming for both authors and testers [53]. Authors have to prepare code, document it, and explicitly render all dependencies on compilers, operating systems, and hardware. For testers, assessing code and data for repeatability can be challenging since documentation is rarely complete and perfect.

Mesirov emphasized on the need of reproducibilityble research systems that make it easy for authors to perform analyses and then embed them directly into a paper [30]. `SOLE` is a step towards the realization of such a combined environment: It targets authors for whom experimentation and writing research papers continue to remain separate activities, but who would also like a less burdensome yet efficient mechanism to associate their research papers *post-hoc* with the inputs and outputs of the scientific process. `SOLE` allows both readers and reviewers to see more detailed descriptions beyond those described in the paper and to repeat experiments to check assumptions, robustness, and validity of data.

2.3 The `SOLE` Framework

The `SOLE` framework [44, 55] assists authors to present their publications in a descriptive and interactive manner to readers and reviewers. The framework consists of tools that aid authors to render their computational research in a form amenable to linking with a publication and for use by readers and reviewers for assessment and verification. In this section, we describe the framework and its elements. We present first its architectural components and then the tools that authors can use to create science objects and to present their publication in a descriptive and interactive manner.

2.3.1 Publication and Purpose

We are interested in enabling the capture of research studies that process and combine large, diverse longitudinal, complex and distributed data using computing to generate results. We assume that such studies are primarily disseminated through conference and journal publications that typically have a page limit, thereby limiting descriptions in a publication. The investigators who perform such studies are the primary authors of these publications, which are assessed and verified by readers from the wider community.

Authors typically use the document metaphor in which descriptions and claims about the chosen scientific method are made with the help of textual elements, such as text, figures, tables, images, plots, and bibliographic links. A reader as a first step is interested in learning and understanding the scientific method, but soon graduates to reading the publication with a purpose, which may vary based on their need and role (See Table 2.1). For instance, some readers may want to repeat experiments for reviewing purposes to verify the author's claim that the experiments are correct. Alternatively, if an author claims development of a software package, readers may want to reuse that package in their own research.

TABLE 2.1: Reader purposes for evaluating a scientific publication

Purpose	**Meaning**
Reference	Be able to reference data, methods, and processes at various granularities.
Redescribe	Annotate the publication text with further descriptions.
Repeat	Execute the processes followed in the original publication in the same execution order and under the same environment as
Rework	Execute the processes followed in the original publication in the same execution order and under the same environment but with different parameters.
Reuse	Reuse and share data, method and processes or any constituent part of it.
Reproduce	Repeat but with different data, method, hardware, environment, etc.
Review	Be able to audit and validate results.

2.3.2 Science Objects

The SOLE system provides authors with tools that turn data, methods, and processes used in a scientific method to granular units of knowledge, termed as *science objects*. Science objects are self-contained— they contain the content of a data, method or process and metadata, such as the Uniform Resource Identifier (URI) related to the content. The metadata and content of a science object may vary depending upon whether the object refers to a data, method or process. The objective is to use tools to build complete science objects so that they satisfy one or more of readers' purposes as well as contain information that makes them amenable to be linked or embedded with a paper.

SOLE tools can be used to create two types of science objects: descriptive and interactive. Descriptive science objects are static and self-contained, in that the content in them is sufficient for a reader to satisfy their purpose. Such objects include annotations, source code fragments, provenance records, database records from a variety of sources, and software packages. Interactive science objects allow a reader to satisfy the purpose through execution of software. The objects have associated metadata that describe how to execute the software and are also aided with an accompanying infrastructure for execution. Examples of such objects are workflows, and virtual images.

Prior to using a SOLE tool, the author must define the scope of the science object. This is achieved through a user-defined *tag*. Once a scope is defined, a SOLE tool can automatically populate the metadata and content of a science object. In general, to scope a science object in SOLE, the author declares a tag with the following syntax:

begin type $name_1|\ldots|name_n$

[science object content]
end

in which *begin* and *end* delimit the tag, *type* defines the kind of science object to create, and $name_1$ to $name_n$ are user-defined names. Thus, the same object can be tagged by more than one name. `SOLE`'s tools process the declared tag and based on each tag's *type* definition creates a science object. The tools also associate a set of metadata elements representing the object, including a reference to the object as a URI. The exact location where the tag should be placed depends upon the type of science object. In the following list we describe the science objects in `SOLE`, where to place the tag and provide an overview of how the tools process the tags (we will describe the tools in detail in Section 2.4).

`SOLE` supports five kinds of science objects:

Language objects. To create language objects, the author places the tags in source code files. An example of such a tag is described in Section 2.4.1. The `SOLE` tool uses Ctags [8] to create the language objects. in a file, but appends a URI to the language objects. We have expanded the Ctags utility to allow for user-defined tags as described above and to include more than one program functions. This allows for linking an algorithm in a paper with multiple functions and data structures defined in multiple files.

Data Objects. To create data objects, the author places tags on PDF files or on the download history, if a dataset is downloaded from the Web. Examples of such tags are described in Section 2.4.2. `SOLE` uses the Poppler library [47] to extract tagged annotations from PDF. The metadata of the tagged annotation includes the URI of the PDF, the exact location in the PDF where annotation was made, and the annotated text. `SOLE` uses the Places schema of the Firefox browser to generate a linkable download history.

Workflow Objects. To create workflow objects, the author places tags on functions in the source file by inserting tags with the workflow *tag_type*. Inputs, outputs and function definitions are determined through the *tag_type*. These workflow objects are defined in Section 2.4.3. `SOLE` tool creates a workflow object that conforms to the Galaxy-ES workflow systems. Galaxy-ES provides an open, web-based platform for specifying tools and running computational experiments as workflows [17]. Each tagged function is automatically wrapped as an appropriate Galaxy tool definition and hosted on the web-server instance connected with Galaxy. Authors can further specify if web services should accept user specified parameters and types of data.

Software Package Objects. No tags are required to create software package objects.The `SOLE` tool uses CDE for creating the package, but wraps the package with user-supplied tag and the resulting metadata annotated software package object is uploaded to the database (Section 2.4.4).

Virtual Image Objects. There are no tags required to create a virtual image objects. Tags are, however, placed in a readable file in the virtual machine. SOLE tools reads the file to export information about the virtual image and create a virtual image object containing necessary metadata about the image and a URI that allows secure connection to the image (Section 2.4.5).

In addition to the above described science objects, SOLE creates **replayable packages** (Section 2.5), which allow an author to assemble code, data, environment, and a reference execution trace into a single package that can be distributed easily, for example to testers who want to re-execute the full source code. Readers and testers can view the execution

trace and specify, at a process and file level, nodes of the graph that they want to re-execute, thus saving time and effort for repeatability testing. This SOLE tool internally depends upon CDE [19, 18] a utility for creating x86 Linux-based packages that requires no installation, configuration or root permissions. It is also described in detail in the chapter by Guo *et. al.* We have further extended the CDE utility to store a provenance graph at the process, and file-level representing a reference run-time execution, which the readers can view and replay.

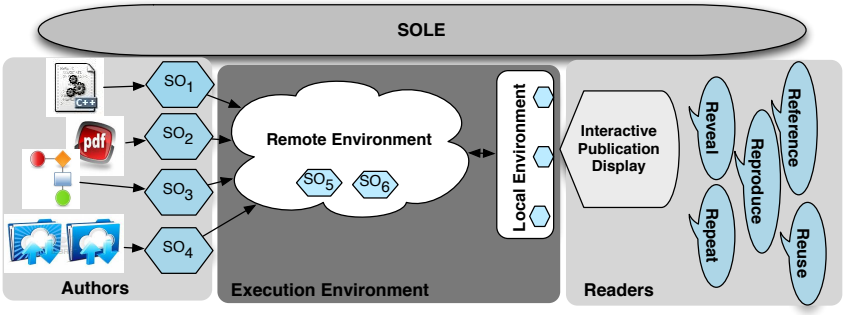


FIGURE 2.1: The SOLE Architecture. Tools create and upload science objects to a SOLE environment, which consists of a metadata repository used to store information about science objects and an execution environment used for interactive science objects. The metadata repository acts as a web-based bibliography, indexed by keywords, which can be used by authors in their authoring environment to associate phrases with objects.

2.3.3 The Environment

The SOLE environment hosts the publication and the science objects. The publication is rendered as a Web page; The science object specification is stored in FluidInfo [12], a lightweight key-value data store, with a complete set of permissions to give users full control over their objects and tags. Depending on the type of science object, its content is either stored in the data store as a value, or the value is a URI that points to a file in the execution environment. Both the storage and the execution environment is currently on the Amazon cloud.

SOLE provides a simple query interface that allows users to search for science objects through their tag names. Since the same tag name may be associated with multiple science objects, the reader must first choose amongst the available options. SOLE provides the reader with a succinct description of the science object in order for the reader to make this choice. Once a science object is retrieved, the author chooses values, phrases, figures or tables within the publication content and links those chosen elements to the science object by clicking on "Attach SO". Internally, this links the chosen element with the science object in a manner similar to a citation being linked to a bibliographic entry.

The environment supports some additional features, namely the bibliography specification of all science objects can be exported to CSV and XML. The authoring environment can be HTML or Word. The environment currently does not support creation or search of user-defined objects. It also does not support collaborative use, though a commenting feature allows many readers to provide feedback on the science object regarding for example, its intended purpose and ability to serve that purpose. A more interactive user-interface is part of our future work.

Figure 2.1 shows the SOLE architecture. Tools create and upload science objects to a SOLE environment, which consists of a metadata repository used to store information about science objects and an execution environment used for interactive science objects. The metadata repository acts as a web-based bibliography, indexed by tag names, which can be used by authors in their authoring environment to associate text phrases and figures with science objects. Figure 2.2 shows the SOLE system in action through an example of a language object.

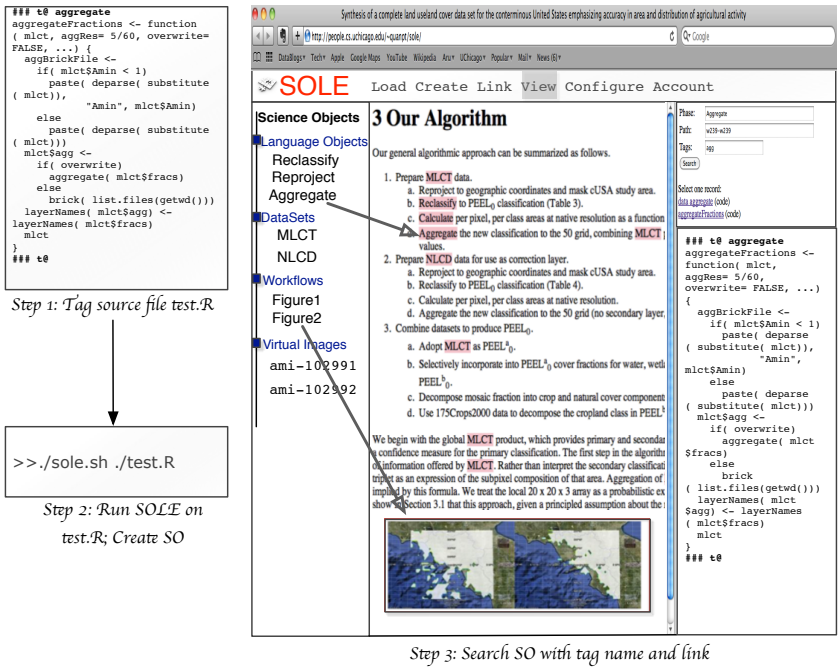


FIGURE 2.2: Figure 2.2 demonstrates steps that an author follows to create and associate science objects in SOLE. The author first identifies the phrase e.g., "Aggregate" in the publication text that would be better understood, assessed and/or verified by the reader if augmented. The author then tags the corresponding source code, runs a SOLE command line tool to create the necessary metadata, and finally associates the phrase "Aggregate' in the paper by recalling it with its tag through a queryable interface. The reader views the implementation of the aggregate function and understands the merit.

2.4 Using Science Objects to Improve Reproducibility in Papers

We use published examples from climate science, computer science, and biochemistry to demonstrate the use of science objects. Our climate science and computer science examples are from publications that are part of the Center for Robust Decision making on Climate and Energy Policy (RDCEP), a collaborative, multi-institutional project that aims to improve the computational models needed to evaluate climate and energy policies and to make robust decisions based on outcomes [63]. In RDCEP, sharing science objects in the form of

data, tools, and software is critical; it enables scientists to compare models and to build more accurate models. Currently in RDCEP science objects are shared through a companion web site. We show here how to construct these science objects and link them with RDCEP publications using SOLE. We have also taken biochemistry publications [46] that use PubChem [64] datasets to demonstrate the use of annotated dataset science objects.

2.4.1 Language Objects

Computational research papers abstract execution and source code and typically convey computational concepts through equations, methods, pseudo-code or algorithms. While such abstractions are necessary to communicate with the reader at a conceptual level, they can hide vital details that are present in the source code. For verification of the text, these details must be made available along with the conceptual description [32].

For instance, Cai et. al. [6] describe a numerical dynamic program for multi-stage decision making problems. The paper describes the pseudo code for the numerical dynamic program but does not provide implementation details of the program, such as whether floats or ints are used for approximation or whether the program uses a top-down memoization approach or a bottom-up method for solving sub-problems. Such details if present along with the algorithmic description can be vital for a reader who plans to apply the algorithm on large amounts of data since a bottom up approach has no overhead for recursion.

Similarly interdisciplinary research papers on business and industrial engineering that use monte-carlo simulation methods to decide about risk and uncertainty often ignore crucial details such as the chosen probability of distribution on its inputs. In business domains, it is typical to use a triangular distribution instead of a uniform distribution, which is more common in industrial engineering. However, unless this finer detail is associated with the description of monte-carlo methods in the paper, it is difficult to quickly verify if the paper has adopted known knowledge.

SOLE's language science objects improve overall readability and understanding of concepts described in a paper by annotating them with details present in source codes. Two approaches for annotation have been explored in previous work. In the first approach, the concept and the source code are mapped to a pre-defined ontology class. For instance, an initialization step and the configuration file will map to an initialization class. This approach is demonstrated in ontologies such as MGED [67] and EXPO [54] that allow for the linking of experimental descriptions with the resources used. In the second approach, there are no predefined vocabulary classes, but a language object is defined on the fly by taking a fragment from the source code and associating it with the algorithmic description in the paper. In SOLE, we follow the second approach since it allows greater flexibility.

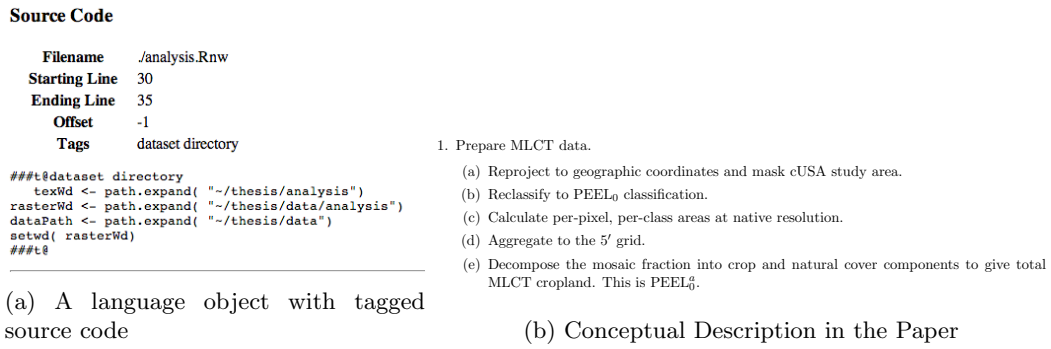
SOLE uses the Ctags [8] and user-defined tags on source files to create language objects on the fly. By default, Ctags generates an index (or tag) file of language objects found in source files that allows these items to be quickly and easily located. A tag signifies a language object for which an index entry is available (or, alternatively, the index entry is created for that object). To identify a language object, Ctags identifies the language of the source code file and calls the appropriate language parser to operate on the file. Whenever the parser finds an interesting token, it calls a function to define a tag entry. Ctags parsers are available for more than forty programming languages. In SOLE, we have modified Ctags parsers for common programming languages, such as C, C++, JAVA, and R. The modified parsers identify both language specific tokens and user-defined tokens (described shortly), which can be present within a language object, span across more than one language token, or across source code files.

The user-defined tokens are declared according to the syntax defined in 2.3.2, in which the words *begin* and *end* are replaced with comment style of the source code, type defines

the type of source code object, and the name declares the arbitrary concept to which the user wants to associate the language object(s). The name is indexed and tagged, and the author can use the tag to associate this user-defined object with the phrases in the text. Figure 2.3a shows an R function that is annotated with user-defined tags. It also shows the accompanying metadata about the function, which is part of the language object.

Demonstration: The Best. et. al. paper on the "Synthesis of a Complete Land Use/Land Cover Dataset for the Conterminous United States" [3] is a computational paper that generates a new land cover dataset for the conterminous USA called "$\texttt{PEEL}_0$". The paper proposes novel synthesis methodology, which combines information from multiple sources by establishing a common classification scheme at lower spatial resolution. It shows how to generate the dataset by combining data from existing data products the MODIS Land Cover Type (MLCT) and the National Land Cover Database (NLCD).

In the paper, the synthesis methodology, which is a 3-step process is described conceptually. For lack of space, one step of the methodology is shown in Figure 2.3b. Please refer to the paper [3] for the other two steps. Finer algorithmic details, such as the specific R functions used for re-projection or reclassification are not described. By attaching a language object, the author can make the definition of these terms precise and remove ambiguities in interpretation. Annotated science objects from this paper are demonstrated in the `SOLE` system [56] and linked to specific text phrases.



Source Code

Filename	./analysis.Rnw
Starting Line	30
Ending Line	35
Offset	-1
Tags	dataset directory

```
###t@dataset directory
   texWd <- path.expand( "~/thesis/analysis")
rasterWd <- path.expand( "~/thesis/data/analysis")
dataPath <- path.expand( "~/thesis/data")
setwd( rasterWd)
###t@
```

1. Prepare MLCT data.
   (a) Reproject to geographic coordinates and mask cUSA study area.
   (b) Reclassify to PEEL_0 classification.
   (c) Calculate per-pixel, per-class areas at native resolution.
   (d) Aggregate to the 5′ grid.
   (e) Decompose the mosaic fraction into crop and natural cover components to give total MLCT cropland. This is PEEL_0^a.

(a) A language object with tagged source code

(b) Conceptual Description in the Paper

FIGURE 2.3: Enhancing conceptual description with Language Science Objects

2.4.2 Data Objects

Research papers commonly cite data through a bibliographic reference or through a digital object identifier (DOI). The latter approach provides a permanent identifier for the data, even if the location of data or its metadata change over time. While these citation mechanisms are necessary for linking datasets they may be insufficient for the purpose of repeatability since they do not specify which part of a dataset is used in the research paper. For instance simply referencing a report that is 100 pages long often does not help. A reviewer verifying values or tables taken from the report must know the references at the granularity of pages and paragraphs.

The problem of coarse linking of datasets is acute when the web link refers to an online database such as PubChem. This database allows users to access subsets of data (e.g., molecules) from its repository through Web forms and query interfaces. Accessed datasets have a scientific import for research, but are commonly cited using the web address of the primary repository. Research papers may provide text descriptions of how the dataset was

downloaded from the primary online database, and include specification of query parameters, such as dates or spatial region clauses. In a few cases publications provide a link to the downloaded dataset. In neither case can the reviewer verify how the dataset was derived from the database, with the specification often falling short. Reviewers often have to re-learn the mechanism for downloading the dataset as used in the publication. It would be helpful if the downloaded dataset could be identified with a link or a DOI that is derived from the primary database's link or DOI and that also includes metadata, such as query parameters that describes how the dataset was obtained.

An important direction towards citing data at a finer granularity is the Open Archives Initiative Object Reuse and Exchange (OAI-ORE) project, which exposes the rich content in Web resources to support their reuse in authoring, exchange, and preservation. Web resources are considered as compound digital objects consisting of many distributed resources with multiple media types including text, images, data, and video. The project goal is to expose the rich content in these aggregations through a specification, which hopefully may become a standard for describing Web resources.

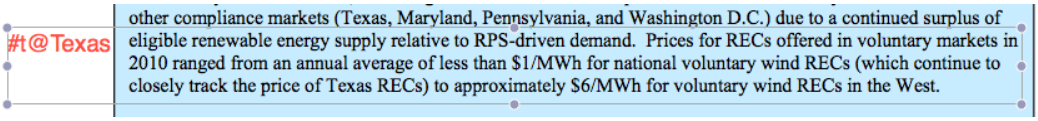
While OAI-ORE provides a useful specification for Web resource citation, the need to supply many metadata values as part of the specification can be cumbersome for an author. Therefore in `SOLE` our primary focus is on tools that can provide input values to the metadata specification in an automated way. `SOLE` provides two tools for automating the metadata extraction problem. For PDF sources, `SOLE` provides, `PDFExtractor`, a tool that extracts segments from a PDF or postscript document and produces a bibliography of the segments. The segments have the DOI of the primary PDF but are enhanced with additional metadata as required by the OAI-ORE specification. If the datasource is an online database, `SOLE` provides `RepAudit`, a browser plugin to silently audit the parameter specification or the link traversal. It provides both the downloaded dataset in the original format and also provides a metadata file, based on OAI-ORE specification with values of the metadata entries. We use examples to illustrate the use of the two tools:

`PDFExtractor` extracts segments, such as text, images that are selected by the user through the annotation command in the PDF reader. Figure 2.4a shows a PDF document with a segment selected by the user. Figure 2.4b shows the resulting science object, indexed through user-defined tags. Metadata in the resulting science object is according to the OAI-ORE specification and enables precise location of the segment within the PDF. Internally the `PDFExtractor` uses the Poppler library [47] to extract the segments and attach metadata specification to the extractions. If the user is connected to a `SOLE` environment, `PDFExtractor` also transmits the bibliographic information to the FluidInfo database.

Demonstration: Authors annotate "Feasibility of U.S. renewable portfolio standards under cost caps and case study for Illinois", which is a policy paper that surveys US state renewable portfolio standards (RPS), cost caps, and studies in particular the feasibility of the RPS proposed for Illinois through a cost model [20]. The paper reports RPS-related data from more than a dozen reports (cited in the references) published by different federal and state energy departments. Often RPS and cost values, for purposes of brevity, are aggregated from their original source and an average is mentioned. It is difficult to verify such a policy paper unless the papers are referenced with page numbers and the approximate location of each value on the page.

We used `SOLE` to annotate the reports cited in this paper [57]. For example, the reference "[13]" mentioned in the excerpt of the paper in Figure 2.4b is one report that is tagged. The tagging is shown in Figure 2.4a. The authors simply annotate the PDF with an easy to remember tag, such as "Texas", and then execute the `PDFExtractor` tool on the saved PDF. The tool generates a corresponding science object with the additional metadata about the

page number in the PDF and approximate location, and also uploads the science objects to the `SOLE` system. The author then attaches these science objects to the text phrases in the paper. The reader can verify that even that the authors have approximated the value of $1 per MWh.



(a) Annotate the PDF by selecting a rectangular region and adding tag *Texas*

RPS was initially met by existing hydropower rather than new construction [6]. Texas (1999) possesses such anomalously strong wind resources that development of windpower in the state could be driven largely by the federal Production Tax Credit and Investment Tax Credits with the state RPS playing a much less significant role. (The Production Tax Credit and Investment Tax Credits, henceforth "PTC", reimburse qualifying renewable generators for up to 30% of the installed cost. See Appendix A.1 for further discussion.) Current Texas REC prices remain so low (∼ $1 per MWh; [13]) that the state RPS is not a significant subsidy for windpower in Texas, and current construction implies that Texas wind capacity will reach its 10 GW target almost fifteen years ahead of RPS-mandated requirements [14]. Prediction of the expected evolution of renewables implementation

Annotations

Filename	wintechnology.pdf
Description	R. Wiser, M. Bolinger, 2010 Wind Technologies Market Report, The U.S. Department of Energy (2011)
Page	54
x	277.844560
y	315.829000
Tags	Texas

$20/MWh by year-end. REC prices to serve RPS requirements in New Jersey, Illinois, and Delaware fell below $5/MWh by the end of 2010, following declines in 2009. REC prices remained at similarly low levels in several other compliance markets (Texas, Maryland, Pennsylvania, and Washington D.C.) due to a continued surplus of eligible renewable energy supply relative to RPS-driven demand. Prices for RECs offered in voluntary markets in 2010 ranged from an annual average of less than $1/MWh for national voluntary wind RECs (which continue to closely track the price of Texas RECs) to approximately $6/MWh for voluntary wind RECs in the West.

High-Price REC Markets Low-Price REC Markets

$80 $20

(b) The extracted text from the paper (top). The highlighted value $1 per MWh refers to the annotation science object with the tag *Texas* (bottom)

FIGURE 2.4: The PDFExtractor Tool

`RepAudit` is a browser plugin for Mozilla Firefox. If enabled by the user before exploring an online database, `RepAudit` first determines the primary link of the online database being explored. If no link can be determined, `RepAudit` assigns a temporary link which can be changed later by the author. `RepAudit` audits user's links and/or the keywords mentioned in query forms and maintains a history of the visited links and pages in a SQLite database. This database is built using the Firefox Places schema, which stores new URls and their pages. The `RepAudit` database also stores link traversal history. The history for a downloaded dataset can be exported as a derived link.

Demonstration: We have experimented with the use of `RepAudit` for various biochemistry publications that use data from PubChem. These publications are provided by PubChem as dataset exemplars. To locate the correct data set as described in the publications, we worked with biochemistry graduate students at our institution. Here we describe the use of `RepAudit` on one of those publications.

The biochemistry paper "Identifying Compound-Target Associations by Combining

Bioactivity Profile Similarity Search and Public Databases Mining" [7] provides the following dataset description:

> The NCI-60 data set contains anticancer screening results for more than 40,000 compounds. It is publicly available in the PubChem BioAssay database(38) as 73 bioassays with the name of NCI human tumor cell line growth inhibition assay under the DTP/NCI data source. In this work, only the top 60 bioassays (referred hereafter as NCI-60) with the largest number of tested compounds were selected (Supporting Information, Table S1). Relevant bioactivity data were downloaded at the PubChem FTP site (`ftp://ftp.ncbi.nlm.nih.gov/pubchem/Bioassay`, accessed on December 9, 2010). A total of 5083 compounds were found commonly tested in all of the 60 bioassays. Additional data set characteristics are summarized in Supporting Information.

In collaboration with graduate biology students we searched PubChem for the NCI-60 data set. Even though the description mentions NCI-60 as publicly available from PubChem site, no assays can be retrieved with the specific search term. However, 195 bioassays are indeed retrieved from the search term "NCI human tumor cell line growth inhibition assay" with 75 bioassays corresponding to bioassays under the source DTP/NCI uploaded earlier than December 9, 2010. There is no way of sorting or filtering the bioassays on the largest number of tested compounds from the website from both search and advanced search features. In addition the URL `ftp://ftp.ncbi.nlm.nih.gov/pubchem/Bioassay` provides no search interface or a directory of NCI-60 dataset. Thus it is not entirely evident what specific search terms were used by the authors and the filters applied by them.

In the `SOLE` demonstration [58] we show the PubChem dataset description tagged with the browser history of the downloaded dataset. The history is shown in three representations: (1) as full paths from the beginning of the session to the downloaded dataset in the end, (2) as distinct paths, and (3) as search terms. The obtained history shows that downloading a dataset requires significant data retrieval from PubChem and that a simple URL is not sufficient for obtaining the exact same data.

2.4.3 Workflow Objects

A typical computational paper is a result of several data management and computational tasks performed independently, such as gathering and preparing input data; configuring; building and running the model on various computing resources; collecting metadata about the model; computing environment and input files; post-processing model outputs; and visualization. The paper mostly describes the end results of these tasks. However, the reader would like to change one or more parameters or steps of the task to assess the impact on the result. For a reader, the easiest way to check the workability of the application is when parameters can be changed through command-lines are interfaces, without having to change source codes and recompiling the application. However, substituting one or more steps with another is often more challenging without source code modification.

Workflow systems provide abstractions for designing, composing, and executing applications with multiple steps such that steps may be conveniently substituted and parameters easily modified. When the task is supposed to be performed many times, such abstractions make it easier for the scientist to focus on the logic of their applications instead of the technical details for accessing and invoking the software components needed for execution, which can be delegated to the workflow system. Despite the advantages, most domain-specific workflow systems are often strongly typed necessitating a scientist to abandon their favorite programming environment.

Galaxy-ES is a workflow system for climate science that allows a scientist to remain with

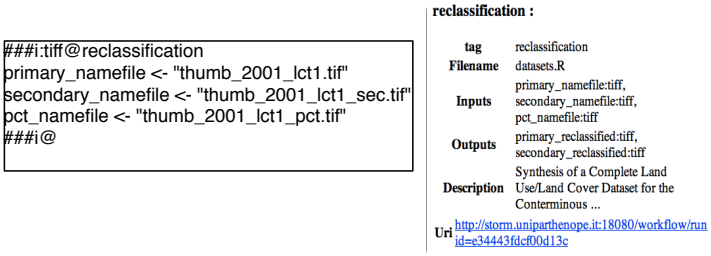


FIGURE 2.5: The tagged source code for workflow inputs (left). The created workflow object (right). Clicking on the workflow executes the workflow

his or her programming environment but provides support for wrapping their application with simple, uniform Web interfaces. An application can be executed as multiple steps, while the system automatically manages the computational details. From these Web interfaces a reader can change parameters to run an application. Galaxy-ES builds from the Galaxy system in genomics, which allows experimentalists without informatics or programming expertise to perform complex large-scale analysis with just a Web browser. Galaxy-ES, similar to Galaxy, also introduces two other features for reproducible research: (1) it automatically tracks and manages data provenance and provides support for capturing the context and intent of computational methods, and (2) Galaxy Pages are interactive, web-based documents that provide users with a medium to communicate a complete computational analysis.

However, to use Galaxy-ES the author still has to provision an instance and build a *tool definition* of their application. In particular, to provision and define a tool an author still has to write three files: (1) a configuration file describing the tool user interface and how the tool have to be executed including inputs and outputs; (2) a launcher script that prepare the custom execution environment and redirect output and error streams to a self describing container; and (3) wrap scripts that perform the dataset staging in, the actual processing and the dataset staging out. These steps do require familiarity with the internals of the Galaxy system and the details associated with tool definitions, which may be significant for an author.

To reduce this gap, we extended the `SOLE` approach of tagging source codes. Authors continue to work in their favorite declarative programming language and tag functions in source codes with the tag definition as described in Section 2.3.2. The tag_type describes if the tagged portion of the code describes inputs to a function, outputs of a function, or the function definition. `SOLEConnect` is a `SOLE` tool that interprets these tag types and creates a corresponding Galaxy-ES tool definition, which are then automatically deployed on a Galaxy instance running on Amazon cloud. `SOLEConnect` also creates a corresponding science object of this tool describing metadata elements such as the URL of the tool, its inputs, outputs and the corresponding language science object. The metadata elements are uploaded to FluidInfo.

Demonstration: To demonstrate the use of `SOLEConnect`, we continue with example of Best et. al. for which we created language objects in Section 2.4.1. The paper describes a novel synthesis methodology to general a land cover dataset for the conterminous USA called "$\texttt{PEEL}_0$". The methodology is a three step process: The first step inputs a dataset; The second step takes numerical parameters as inputs to execute a model on the datasets, and the third step aggregates relevant information from datasets, based on the model to generate a new model with a visualization of the data. In reviewing this methodology, readers are

interested in experimenting with different datasets, changing the parameter values in the second step and/or substituting the model in the second step to see the impact on the aggregation step.

The three steps of the methodology can individually be described as Galaxy-ES tools but require effort from the author's side to wrap them as tools. Instead the steps were tagged with `SOLE` tags.The inputs of the first step are tagged as shown in Figure 2.5. `SOLEConnect` creates a workflow science object (Figure 2.5), which can be attached with description of the first step (Figure 2.3b) in the publication. To experiment with the linked workflow science object in `SOLE` with input fields that allow a reader to choose different datasets, please see [59].

2.4.4 Software Package Objects

In some cases readers may find it useful to attach an entire software package with the paper. CDE is tool that uses system call interposition to automatically create portable software packages. The primary advantage of using CDE is that creating a package is completely automatic, and that running programs within a package re-quires no installation, configuration, or root permissions. The `SOLEPkg` tool uses CDE for creating the package, but wraps the package with user-supplied metadata including a tag. The resulting metadata annotated software package object is uploaded to the database. The author can attach this SO to phrases in the text similar to a language object. In Section 2.5 we show how replayable software packages can be created, which not only help in verification but also help reduce the time to replay.

2.4.5 Virtual Image Objects

Virtualization software now provides a level of maturity wherein a running instance of a computer (a virtual machine image (VMI)) with resources, such as memory and central processing units (CPUs), can be automatically provisioned to perform computations and data management tasks. An immediate impact of this advancement in virtualization technology is on reproducibility of experiments. A scientist can conduct an experiment on a virtual machine and simply share the image of that machine with the reader. By providing an authoritative, encapsulated and executable record of computations, virtualization provides the reader with exactly the same environment as that used by the author.

This use of virtualization has been exploited by many recent projects to enable reproducibile experiments. Typically, it is a two step process. First, scientific datasets are stored in the cloud with more researchers having access to the data. Second, the entire computer system, including the operating system, is copied to the VMI, so that other researchers can fully replicate the in silico experiment. The author states that by providing a pre-installed, pre-configured virtual machine the original laboratory is provided intact for future researchers [11].

While a VMI may provide an authoritative and executable record of computations[1], the computation is encapsulated in a machine that is not directly attached with the descriptions in the paper. It would be useful if there was a general purpose convention for addressing where one should look to understand what a VMI is about, the experiment that it contains and how to modify the experiment.

`VMIReader` is a `SOLE` tool that uses open-source VirtualBox utility for provisioning an AWS VMI. The utility creates a Readme file in the user home directory, which the user

[1]We assume that the image is provisioned on a standard cloud provider. In general guaranteeing executability of a VMI is challenging due to differing standards to encode an image.

can use for connecting phrases in the paper with specific paths in the VMI or executables. The Readme file is synced with the `SOLE` system and the system stores the tags associated with the VMI. Due to size constraints the actual VMI is not stored in `SOLE` but a link corresponding.

Demonstration: To demonstrate the use of `VMIReader` in SOLE we revert to the RDCEP paper on the "Synthesis of a Complete Land Use/Land Cover Dataset for the Conterminous United States" [3], which consists of $\texttt{PEEL}_0$ an R-program for generating land cover dataset for the conterminous USA. $\texttt{PEEL}_0$ combines information from multiple sources by establishing a common classification scheme at lower spatial resolution. Executing $\texttt{PEEL}_0$ would require specific versions of R and libraries: R `v2.15.2`, libgdal `v1.7.3`, raster `v2.0.41`, rgdal `v0.8.5`, ggplot `v2.0.9.3`, xtable `v1.7.0`, and doMC `v1.2.5`.

To share $\texttt{PEEL}_0$, the author can build a software package (with CDE) or build a virtual image preinstalled with R and the accompanying libraries. The advantage of building a VMI is that it can also pre-loaded with the MODIS and National Land Cover Database (NLCD) datasets that were used in the experimentation. However, this VMI is still not does explicitly map the three steps described in the paper with the source code. We created a `SOLE`-specific ReadMe file with the tags "Step1", "Step 2", and "Step3", with each step describing the command-line specification to run the respective step. This ReadMe file is part of $\texttt{PEEL}_0$ source code directory. Now, when the `VMIReader` provisions the virtual machine with the dataset and source code it interprets the `SOLE`-specific ReadMe file and uploads the tags and their specification to the `SOLE` system. The author can attach the tags to the individual steps mentioned in the paper. The `SOLE VMIReader` demonstration is available at [60].

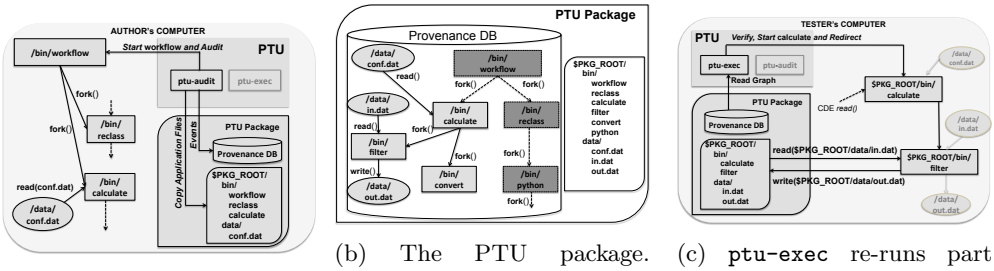
2.5 Science Objects for Repeatability Testing

Increasingly conference committee and journal editors are encouraging authors to submit their code, data, and software for repeatability testing. Repeatability testing improves peer review by allowing reviewers to: (1) not only read the ideas in the paper, but validate them by running the accompanying software; (2) run the software for different data and parameters to check robustness; and (3) determine the limitations and assumptions in the ideas by testing with more general inputs, or under different conditions and environments.

However, as documented by recent experiments, repeatability testing can be arduous and time consuming for both authors and testers [28, 13]. Authors have to prepare code, document it, and make explicit rendering of all dependencies on compilers, operating systems and hardware. For testers, assessing code and data for repeatability can be challenging since documentation is rarely complete and perfect. But more so, as experiments become data and computation intensive, the test- ing time can be significant [53].

Recently tools have emerged that aid authors and testers in making their software and thus experiments repeatable. CDE helps authors package the code, data, and environment for Linux programs so that they can be run and deployed on other machines without installation or configuration [18, 19]. The resulting x86 Linux-based package requires no installation, configuration, or root permissions. Being open-source, with a user-friendly interface for packaging and rerunning, CDE has become useful for conducting reproducible research.

While CDE is a step towards simplifying repeat- ability testing for authors, they do not reduce computation and data processing time for repeatability tests. Testers may want to repeat only selected portions of an experiment without having to go through the (often time-consuming) process of repeating the experiment in its entirety. For instance, they



(a) `ptu-audit` builds a package of provenance events and copies application files.

(b) The PTU package. The tester chooses to run the sub-graph rooted at */bin/calculate*.

(c) `ptu-exec` re-runs part of the application from */bin/calculate*. It uses CDE to re-route file dependencies.

FIGURE 2.6: The Provenance-to-Use (`PTU`) Tool

may want to avoid running a compute-intensive process that decodes and splits an MPEG-video, or avoid performing a data-intensive text scan, or avoid network communication or transfers. Similarly, they may want to reuse cached results. However, such selective execution of program components is difficult or impossible for the tester, unless appropriate history and provenance information has been captured in a previous reference run.

Provenance-To-Use (`PTU`) [43, 50] is a `SOLE` tool for reducing time for repeatability testing. Authors can use `PTU` to accomplish two tasks: (1) build a package of their source code, data, and environment variables, and (2) store process and file level details about a reference execution of their system in an accompanying database. A package alleviates testers from software deployment issues allowing for convenient distribution. Recording a reference execution path and run- time details within the package eases distribution of this vital information that can guide testers during the test- ing phase. In particular, testers can explore the provenance graph and accompanying run-time details to specify the part of the provenance graph that they want to re-execute or replay: see Figures 2.6a, 2.6b, and 2.6c.

`PTU` uses CDE [18, 19] to create and run a package. CDE uses the Unix *ptrace* system call interposition to collect code, data files, and environment variables. The *ptrace* mechanism also allows for auditing file and process information, which can be transformed for storing a provenance graph, independent of the application [34]. In `PTU`, we enhance CDE's use of *ptrace* to store a provenance graph representing a reference run-time execution at the process and file level. We describe how `PTU` can be used by authors and testers.

Authors use PTU to create a self-contained package with a reference execution by prepending the application command with the `ptu-audit` tool as in the following example, which involves the Java application *TextAnalyzer* applied to a file *news.txt*:

ptu-audit `java` *TextAnalyzer news.txt*

The `ptu-audit` tool uses *ptrace* to monitor 50 system calls including process system calls, such as $execve(), sys\_fork()$; file system calls, such as $read(), write(), sys\_io()$ for collecting file provenance; and network calls, such as $bind(), connect(), socket()$, and $send()$ for auditing network activity. Whenever system calls occur, `PTU` notes the identifier of the process that made the system call to extract more information about the process from the Linux /proc file system. In particular, the following information is obtained: processs name, owner, group, parent, host, creation time, command line, environment variables, and a file's name, path, host, size, and modification time. A separate thread is used to obtain memory footprint, CPU consumption, and I/Os of the process from */proc/*$*pid/stat* every three seconds.

In the case of distributed applications involving multiple compute nodes, network activity is audited independently at each node using *ptrace*, i.e., without coordination with the other node. Since network system calls are blocked during auditing, the process record is always present and the information extracted will be current and accurate. Provenance information about processes and files is stored in the form of an Open Provenance Model (OPM) [31] compliant provenance graph in a SQLite database. Currently, PTUs provenance collector makes two assumptions. First, only network connection information is audited and no network dumps are made. Thus, while a tester can replay a computation between exactly the same set of hosts they cannot do so without conducting network communication. Second, PTU does not audit non-deterministic functions such as *ctime()* and *random()*. Relaxing these assumptions to build a general, distributed audit system is part of our ongoing work.

When a system call (file or network) returns, and if a new file or network does not exist, `PTU` emulates CDE functionality. In the case of a file, it copies the accessed file into a package directory that consists of all sub-directories and symbolic links to the original files location. In the case of network communication, it saves a log of the network connection information, including IP and port information, ordered by time. The package directory also contains the SQLite database that stores the provenance information of the test run.

When the entire reference run finishes, `PTU` builds a reference execution file consisting of the topological sort of the provenance graph. The nodes of the graph enumerate run-time details, such as process memory consumption, and file sizes. The tester, as described next, can utilize the database and the graph for efficient re-execution.

Testers can examine the provenance graph contained in a package to study the accompanying reference execution. This graph can be viewed at the granularity of processes and files, and can aid the tester in visually determining parts of the program that they wish to re-execute. For processes, an accompanying bar graph shows CPU and memory consumption. A tester can then request a re-execution, either by specifying nodes in the provenance graph or by modifying a run configuration file that is included in the package. The configuration file initially specifies the provenance graph, corresponding to the reference execution, ordered topologically. A tester can turn flags on or off for each process and file in the provenance graph, to specify if the process needs to be run or if the file needs to be re-generated, respectively.

To re-run the package, testers prepend the program command with a ptu-exec tool as follows:

ptu-exec `java` *TextAnalyzer news.txt*

The `ptu-exec` reads the provenance graph/run configuration file, and determines if any additional process needs to be run or files must be re-generated. A re-run of a process or a regeneration of a file is mandatory if: (1) A process/file is in the descendent sub-graph of another process that is marked for re-running; (2) A process/file is in the ancestor sub-graph of another file that is marked for regeneration; or Re-running these additional processes and regenerating the files is necessary to maintain consistency of the provenance that will be obtained from the test run.

To re-execute, `ptu-exec` obtains run configuration and environment variables for each process from the SQLite database. To re-execute a process, ptu-exec again monitors it via *ptrace* and re-executes CDE functionality of replacing path argument(s) to refer to the corresponding path within the package *cde-package/cde-root/*. By doing so, cde-exec creates a chroot-like sandbox that tricks the target program into believing that it is executing on

the original machine [18]. The following are two of the several `ptu-exec` command-line options that allow testers to control testing.
-time -t1 <t_1> **-t2** <t_2> implies the tester can re-execute everything between t_1 and t_2. If t_1 is null or t_2 is null execution is done from beginning to t_2 or t_1 till end. With this option the run configuration file need not be modified.
-input <p> <f_1> <f_2> allows user to specify input f_1 instead of f_2 for process specified with process id <p>. This helps user test the workability of the system.

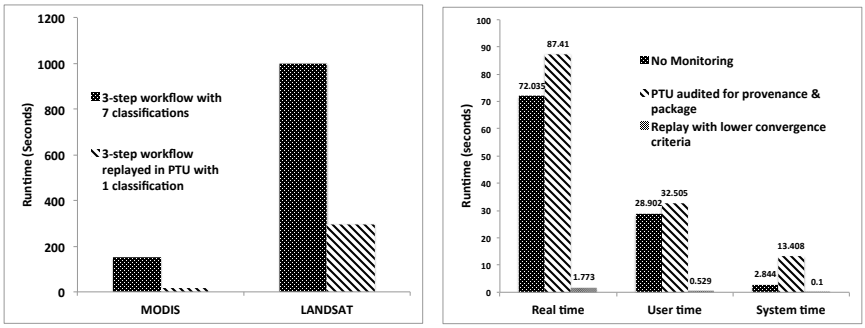
A key distinguishing characteristic of our approach is that it can provide fast re-execution and correct output even when no workflow definition or management system is in place. `PTU` does not force testers to keep mental track of processes that were run, while allowing them to use simple command-line options. Our design does assume that provenance metadata is complete and does not omit, alter, or create false provenance events, and is stored persistently.

Demonstration: To test `PTU` we took two papers [3, 35] in which authors shared data, tools and software. We constructed meaningful testing scenarios and determined if `PTU` provides any performance improvements. The first paper is again Best et. al. that describes the synthesis methodology to generate a new landcover dataset for the conterminous USA called $\texttt{PEEL}_0$. The methodology is implemented as an R-program and combines information from multiple sources by establishing a common classification scheme at lower spatial resolution. This is a three step workflow process, in which the second step is to execute a model which inputs among other parameters a classification scheme. This step is memory-intensive and a typical run takes close to a GB of real memory. To reduce memory consumption, testers may want to further lower the spatial resolution by specifying even fewer input files and a simpler classification model.

Our second program is a text analysis program that mines newspaper articles obtained freely via the Internet to ask questions about water management issues. The program is based on the Java-based Unstructured Information Management Architecture (UIMA) and runs a named-entity recognition analysis program using several data dictionaries. The process splits the input file into multiple input files and runs a parallel analysis on all input files. The tester may want to rerun the program on one input split, but with higher convergence criteria to manually verify the correctness of the program. Note that in general testing these programs will require R and UIMA to be installed, but by using CDE, testers are able to test without any pre-requirements.

The graphs generated at the process and file level in both programs are big. $\texttt{PEEL}_0$ generates a provenance graph with 5 process nodes, and 10000 exclusive file reads based on the number of files in the dataset, and 422 exclusive file writes for the aggregated dataset. In TextAnalyzer, a single run of the job generates a provenance graph of 8 process nodes, total conducting 616 exclusive file reads, 124 exclusive file writes, and 50 file nodes that are read and written again. To improve the readability of these graphs, a single process view is shown that shows just the subgraph corresponding to that process–the files that it reads and the files that it writes. The tester notes the process identifiers of the start process for replay which in case of both $\texttt{PEEL}_0$ and TextAnalyzer continues to remain as the first process. The tester also writes a configuration file in which different inputs are specified for selected processes: in the case of $\texttt{PEEL}_0$ input to the first process is a changed directory location pointing to the new set of files, and for the second process it is the location of the file specifying a simplified classification scheme; in the case of TextAnalyzer it is set of process identifiers to reexecute with a different parameter value. The provenance graphs and the configuration files for the programs can be viewed at [49, 48].

Figure 2.7a show the performance improvements of using `PTU` with $\texttt{PEEL}_0$, and Figure 2.7b show the performance improvement of using `PTU` with TextAnalyzer. The most im-



(a) Time reduction in testing $\texttt{PEEL}_0$ using `PTU`

(b) Time reduction in testing TextAnalyzer using `PTU`

FIGURE 2.7: Using `PTU` to reduce time for repeatability

portant statistic is the slow down in running the process during a test execution at about 35% for $\texttt{PEEL}_0$ and about 15% for TextAnalyzer. This slowdown is due to additional system call tracing, with many file system calls traced for $\texttt{PEEL}_0$. Both these times are easily offset during the re-execution phase. TextAnalyzer has a significant improvement (>98%) since the entire process is run on a much smaller file.

2.6 Discussion

Research ideas are often non-trivial and complex. Evaluating a non-trivial idea completely is often beyond the time budget of any single paper as this requires running many benchmarks, models, competing solutions. Consequently research publications explore novel ideas but without the associated rigor of experimentation. Given increased competition in research fields, we believe that lack of rigor will continue to persist creating an author-reader divide where an author presents a biased evaluation but a reader cannot completely verify the merit of the idea.

In `SOLE` we have aimed to minimize this divide by creating a layer of tools and interfaces which allows authors to share their computational artifacts and readers to verify them through interfaces. For the greater part, the system does assume that authors and readers have the necessary time and incentives to share and verify. Repeatability testing for conferences presents an interesting case in this dimension where authors and readers have sufficient incentives through the conference system, but limited time to share and verify. We believe the both `SOLE` tools and `PTU` are important research contributions in that direction although for lack of measures it is difficult to quantify the cost of reproducibility.

The cost of reproducibility also raises the question of which papers to reproduce. In `SOLE` we have assumed that this is decided by a mutual consensus between co-authors and readers. In general, aggregating a universal consensus on publications that must be reproduced is hard. However, we do believe that funding agencies can play a pivotal role in this decision since they are the primary sponsors of the research and also a proxy for the interests of the general public.

In `SOLE` we have assumed that papers to be reproduced are published in open-access journals such that there are no copyright issues. This assumption does not apply to a large body of scientific work, which is published in copyrighted journals. Such journals, prior

to publishing, require the authors to relinquish their ownership rights to articles, including copyright [61]. Thus authors have minimal say in access of the publication and how to disseminate its content. By increasing awareness about the importance of reproducible computational research [68], and proposing newer standards such as the Reproducible Research Standard [61], this trend seems to be changing now, with some journals, such as *Nature* [37] and *Biostatistics* [5] beginning to require that code or data be released as a precondition for publication.

In `SOLE` our focus has been towards a general approach to reproducibility. However, making a publication reproducible has indeed required us to understand concepts in climate science and biochemistry. Minimizing the involvement of the human-in-the-loop is one of our goals. We believe that we will have a better understanding of this challenging problem as publications, software, and tools from other disciplines become accessible to use and we use `SOLE` tools to make them reproducible.

2.7 Conclusion

Static text-based publications summarize and market the ideas of their authors , but do not provide access to the basis for those ideas, which are embedded in a collection of data and computational methods. Many scientific communities use the Web to share data and software packages. However, such Web resources remain largely disconnected from the claims made in papers. Readers and reviewers who want to reuse data and methods, assess the validity of findings, and verify results cannot do so. In this chapter, we have introduced and described `SOLE`, a a framework for creating descriptive and interactive publications by linking them with the associated science objects. We described a suite of tools, namely the extended Ctags, `PDFExtractor`, `RepAudit`, `SOLEConnect`, and `VMIReader` that assist the author to create science objects for source codes, datasets and online databases, workflows, and virtual machine images. By hosting them on an online database allows both authors and readers to interactively fetch metadata description about the objects and link them with research papers for the purpose of assessment, repeatability, and verification of research. We believe `SOLE` is a step toward a collaborative environment that engages authors and readers towards reproducible research.

2.8 Acknowledgements

We thank Neil Best for the insightful discussions on conducting reproducible research. We thank David Blair for help with PubChem repository. This work was supported in part by the Center for Robust Decision making on Climate and Energy Policy, under NSF grant number 0951576. The work was performed by contractors of the US Government under contract number DE-AC02-06CH11357.

Bibliography

[1] I. Altintas, O. Barney, and E. Jaeger-Frank. Provenance collection support in the Kepler scientific workflow system. *Provenance and annotation of data*, pages 118–132, 2006.

[2] T. Attwood, D. Kell, P. McDermott, J. Marsh, S. Pettifer, and D. Thorne. Utopia documents: Linking Scholarly Literature with Research Data. *Bioinformatics*, 26(18):568–574, 2010.

[3] N. Best, J. Elliott, and I. Foster. Synthesis of a Complete Land Use/Land Cover Dataset for the Conterminous United States. *SSRN eLibrary*, 2012.

[4] BioMed Central. `http://www.biomedcentral.com/`.

[5] Biostatistics editorial. `http://biostatistics.oxfordjournals.org/content/10/3/405.full.pdf+html`.

[6] Y. Cai, K. Judd, and T. Lontzek. Shape-preserving dynamic programming. *Mathematical Methods of Operations Research*, 2012.

[7] T. Cheng, Q. Li, Y. Wang, and S. Bryant. Identifying compound-target associations by combining bioactivity profile similarity search and public databases mining. *Journal of Chemical Information and Modeling*, 51(9):2440–2448, 2011.

[8] Ctags. `http://ctags.sourceforge.net/ctags.html`.

[9] DeXY. `http://www.dexy.it/`.

[10] D. Donoho, A. Maleki, I. Rahman, M. Shahram, and V. Stodden. Reproducible Research in Computational Harmonic Analysis. *Computing in Science & Engineering*, 11(1):8–18, 2009.

[11] J. Dudley and A. Butte. In silico research in the era of cloud computing. *Nature Biotechnology*, pages 1181–1185, 2010.

[12] FluidInfo. `http://fluidinfo.com/`.

[13] J. Freire, P. Bonnet, and D. Shasha. Computational reproducibility: state-of-the-art, challenges, and database research opportunities. In *Proceedings of the 2012 international conference on Management of Data*, pages 593–596, 2012.

[14] M. Gavish and D. Donoho. A Universal Identifier for Computational Results. *Procedia Computer Science*, 4, 2011.

[15] Giga Science. `http://www.gigasciencejournal.com/`.

[16] C. Goble and D. De Roure. myExperiment: social networking for workflow-using e-scientists. In *Proceedings of the 2nd Workshop on Workflows in support of Large-scale Science*, pages 1–2. ACM, 2007.

[17] J. Goecks, A. Nekrutenko, J. Taylor, and The Galaxy Team. Galaxy: A comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol*, 11(8):R86, 2010.

[18] P. Guo. CDE: Run any Linux application on-demand without installation. In *Proceedings of the 2011 USENIX Large Installation System Administration Conference*, LISA '11. USENIX Association, 2011.

[19] P. Guo and E. Dawson. CDE: Using system call interposition to automatically create portable software packages. In *Proceedings of the 2011 USENIX Annual Technical Conference (short paper)*, USENIX '11. USENIX Association, 2011.

[20] S. Johnson and E. Moyer. Feasibility of U.S. Renewable Portfolio Standards Under Cost Caps and Case Study for Illinois. *SSRN eLibrary*, 2012.

[21] G. Klimeck, M. McLennan, S. Brophy, G. Adams, and M. Lundstrom. NanoHUB.org: Advancing Education and Research in Nanotechnology. *Computing in Science & Engineering*, 10(5):17–23, 2008.

[22] D. Knuth. Literate Programming. *The Computer Journal*, 27, 1984.

[23] D. Koop, E. Santos, P. Mates, H. Vo, P. Bonnet, B. Bauer, B. Surer, M. Troyer, D. Williams, J. Tohline, J. Freire, and C Silva. A Provenance-Based Infrastructure to Support the Life Cycle of Executable Papers. *Procedia Computer Science*, 4(0), 2011.

[24] C. Lagoze and H. Van de Sompel. The Open Archives Initiative: Building a Low-barrier Interoperability Framework. In *Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries*, pages 54–62, 2001.

[25] C. Lagoze and H. Van de Sompel. The Making of the Open Archives Initiative Protocol for Metadata Harvesting. *Library Hi Tech*, 21(2):118–128, 2003.

[26] F. Leisch. Dynamic Generation of Statistical Reports Using Literate Data Analysis. In *Proceedings of Computational Statistics*, 2002.

[27] F. Leisch. Sweave, Part I: Mixing R and LaTeX. *R News*, 2(3):28–31, 2002.

[28] S. Manegold, I. Manolescu, L. Afanasiev, J. Feng, G. Gou, M. Hadjieleftheriou, S. Harizopoulos, P. Kalnis, K. Karanasos, D. Laurent, et al. Repeatability & workability evaluation of SIGMOD 2009. *ACM SIGMOD Record*, 38(3):40–43, 2010.

[29] I. Manolescu, L. Afanasiev, A. Arion, J. Dittrich, S. Manegold, N. Polyzotis, K. Schnaitter, P. Senellart, S. Zoupanos, and D. Shasha. The repeatability experiment of SIGMOD 2008. *ACM SIGMOD Record*, 37(1):39–45, 2008.

[30] J. Mesirov. Accessible Reproducible Research. *Science*, 327(5964):415–416, 2010.

[31] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, et al. The Open Provenance Model Core Specification (v1. 1). *Future Generation Computer Systems*, 27(6):743–756, 2011.

[32] A. Morin, J. Urban, P. D. Adams, I. Foster, A. Sali, D. Baker, and P. Sliz. Shining light into black boxes. *Science*, 336(6078):159–160, 2012.

[33] W. Muller, I. Rojas, A. Eberhart, P. Hasse, and M. Schmidt. ARE: The Author-Review-Execute Environment. *Procedia Computer Science*, 4(0), 2011.

[34] K. Muniswamy-Reddy, D. Holland, U. Braun, and M. Seltzer. Provenance-aware storage systems. In *USENIX Annual Technical Conference*, 2006.

[35] J. Murhy, J. Ozik, and M. Altaweel. Textual Hydraulics: Mining Online Newspapers to Detect Physical, Social, and Institutional Water Management Infrastructure. Technical report, Argonne National Lab, 2013.

[36] P. Murray-Rust, C. Neylon, R. Pollock, and J. Wilbanks. Panton Principles, Principles for Open Data in Science, 2010.

[37] Nature Editorial Policies. `www.nature.com/authors/editorial_policies/availability.html`.

[38] P. Nowakowski, E. Ciepiela, D. Harezlak, J. Kocot, M. Kasztelnik, and et. al. The Collage Authoring Environment. *Procedia Computer Science*, 4(0), 2011.

[39] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. Pocock, A. Wipat, et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[40] Open Access. `http://en.wikipedia.org/wiki/Open_access`.

[41] Open Data. `http://en.wikipedia.org/wiki/Open_data`.

[42] Open Source. `http://en.wikipedia.org/wiki/Open_source`.

[43] Q. Pham, T. Malik, and I. Foster. Using Provenance for Repeatability. In *Proceedings of the 5th USENIX Workshop on Theory and Practice of Provenance*, 2012.

[44] Q. Pham, T. Malik, I. Foster, R. de Niro, and Montella R. SOLE: Linking Research Papers with Science Objects –Demonstration. In *Proceedings of the 4th International Workshop on Provenance and Annotation*, 2011.

[45] PLOS One. `http://www.plosone.org`.

[46] PubChem Publications. `http://pubchem.ncbi.nlm.nih.gov/publications.html`.

[47] Poppler. `http://poppler.freedesktop.org/`.

[48] PTU: TextAnalyzer Demonstration. `http://ci.uchicago.edu/SOLE/Demonstrations`.

[49] PTU: $\texttt{PEEL}_0$ Demonstration. `http://ci.uchicago.edu/SOLE/Demonstrations`.

[50] Provenance-to-use (PTU). `http://ci.uchicago.edu/PTU`.

[51] N. Ramsey. Literate Programming Simplified. *Software, IEEE*, 11(5):97–105, 1994.

[52] RunMyCode. `http://www.runmycode.org`.

[53] D. Sasha. Repeatability & Workability for the Software Community: Challenges, Experiences, and the Future. `http://www.cs.utah.edu/~eeide/archive10/slides/shasha.pdf`.

[54] L. Soldatova and R. King. An ontology of scientific experiments. *J. of the Royal Society, Interface / the Royal Society,*, 3(11), 2006.

[55] SOLE: Science Object Linking and Embedding. `http://ci.uchicago.edu/SOLE`.

[56] SOLE: Language Science Object Demonstration. `http://ci.uchicago.edu/SOLE/Demonstrations`.

[57] SOLE: Data Science Object Demonstration for PDFs. `http://ci.uchicago.edu/SOLE/Demonstrations`.

[58] SOLE: Data Science Object Demonstration for Online Databases. `http://ci.uchicago.edu/SOLE/Demonstrations`.

[59] SOLE: Workflow Science Object Demonstration. `http://ci.uchicago.edu/SOLE/Demonstrations`.

[60] SOLE: Virtual Image Science Object Demonstration. `http://ci.uchicago.edu/SOLE/Demonstrations`.

[61] V. Stodden. The Legal Framework for Reproducible Scientific Research: Licensing and Copyright. *Computing in Science & Engineering*, 11(1):35–40, 2009.

[62] V. Stodden, C. Hurlin, and C. Perignon. RunMyCode.Org: A Novel Dissemination and Collaboration Platform for Executing Published Computational Results. In *IEEE eScience Second Workshop on Analyzing and Improving Collaborative eScience with Social Networks (eSoN)*, 2012.

[63] The Center for Robust Decision making on Climate and Energy Policy (RDCEP). `http://www.rdcep.org/`.

[64] The PubChem Project. `http://pubchem.ncbi.nlm.nih.gov//`.

[65] H. Thimbleby. Experiences of Literate Programming Using Cweb (A Variant of Knuth's WEB). *The Computer Journal*, 29(3):201–211, 1986.

[66] P. Van Gorpa and S. Mazanekb. SHARE: a web portal for creating and sharing executable research papers. *Procedia Computer Science*, 4, 2011.

[67] P. Whetzel, H. Parkinson, H. Causton, and et al. The MGED ontology: a resource for semantics-based description of microarray experiments. *Bioinformatics*, 22(7), 2006.

[68] Reproducible Research: Tools and Strategies for Scientific Computing. `http://www.stodden.net/AMP2011/`.

[69] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. Von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde. Swift: Fast, reliable, loosely coupled parallel computation. In *IEEE Congress on Services*, pages 199–206. IEEE, 2007.