Clear and Precise Specification of Ecological Data Management Processes and Dataset Provenance

Leon J. Osterweil, Lori A. Clarke, Aaron M. Ellison, Emery Boose, Rodion Podorozhny, and Alexander Wise

***Abstract*—With the availability of powerful computational and communication systems, scientists now readily access large, complicated derived datasets and build on those results to produce, through further processing, yet other derived datasets of interest. The scientific processes used to create such datasets must be clearly documented so that scientists can evaluate their soundness, reproduce the results, and build upon them in responsible and appropriate ways. Here, we present the concept of an *analytic web*, which defines the scientific processes employed and details the exact application of those processes in creating derived datasets. The work described here is similar to work often referred to as "scientific workflow," but emphasizes the need for a semantically rich, rigorously defined process definition language. We illustrate the information that comprises an analytic web for a scientific process that measures and analyzes the flux of water through a forested watershed. This is a complex and demanding scientific process that illustrates the benefits of using a semantically rich, executable language for defining processes and for supporting automatic creation of process provenance metadata.**

***Note to Practitioners*—The Internet and associated computing capabilities have made it possible for scientists to derive novel datasets through complex processing of existing datasets that may be collected from many locations. But scientists rarely document dataset provenance - the set of processes and a description of how those processes were used - to allow derived datasets to be recreated. Enabling such recreation is an essential part of repeatable science, and thus it is imperative that any dataset generated by scientific computation include provenance metadata, documentation of the precise way in which that dataset was produced. Provenance metadata can help assure that scientists and others understand the value and limitations associated with using that data, but creating provenance metadata is a difficult and time-consuming problem. This paper describes an approach for helping scientists deal with the production and management of their datasets, including the automated generation of provenance metadata. The approach is based on the use of a precisely defined process definition language. The language is relatively clear and easy for scientists to understand, yet it is precise enough to support their control of the application of computing capabilities to the generation of datasets, and is also an aid to the management and understanding of these datasets. This paper illustrates these ideas by providing a case study of a specific problem in ecological dataset production and metadata provenance generation.**

***Index Terms*—Dataset provenance, process definition, scientific workflow.**

Manuscript received March 31, 2008; revised October 13, 2008 and January 19, 2009. First published August 04, 2009; current version published January 08, 2010. This paper was recommended for publication by Associate Editor Y. Yang and Editor Y. Narahari upon evaluation of the reviewers' comments. This work was supported in part by the National Science Foundation under Award CCR-0205575, Award CCR-0427071, and Award IIS-0705772. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

L. J. Osterweil, L. A. Clarke, and A. Wise are with the Department of Computer Science, University of Massachusetts, Amherst, MA 01003 USA (e-mail: ljo@cs.umass.edu; clarke@cs.umass.edu; wise@cs.umass.edu).

A. M. Ellison and E. Boose are with Harvard Forest, Harvard University, Petersham, MA 01366 USA (e-mail: aellison@fas.harvard.edu; boose@fas.harvard.edu).

R. Podorozhny is with Texas State University, San Marcos, TX 78666 USA (e-mail: rp31@txstate.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2009.2021774

I. Introduction

A. The Problem

Modern computation and communication systems have dramatically changed the way in which science is done. These systems enable scientists to work with datasets and to create models of their research subjects that are far larger and more detailed than were possible in the past. Faster computing speeds enable far more ambitious analyses of these models, leading to the production of far greater quantities of derived scientific datasets. Ever faster global networks make these datasets accessible to scientists around the world. While these new computational and communications capabilities have opened up the possibility of exciting new research, they have also led to new challenges and problems. When new datasets are produced by complex processes, they are often promulgated without adequate documentation describing how they were produced. If scientists are to make appropriate use of the datasets produced by others and avoid misuse by inappropriate application of subsequent processing, then it is imperative that they know how such datasets were produced. Indeed, before scientific results can be accepted, they should be reproduced by other scientists, as reproducibility is fundamental to science.

An obvious way to address these challenges is to associate with each dataset, as an *annotation*, a precise description of the dataset. Such annotations, essentially data items that describe data, are called *metadata*. There have already been many calls for the use of metadata, which typically document such details as the date of generation of a dataset, the name of the investigator, and perhaps some specifications of the hardware and software systems used, as well as details of the individual data items (variable name, numerical format, unit of measurement, etc.). We argue that it is necessary to go further. We suggest that an additional type of metadata, *process provenance metadata*, be attached to all datasets, and when necessary to individual data items. The benefits of such process provenance metadata include facilitating reproduction of the data by others, expedited identification of data items and datasets of interest, and better understanding of which forms of subsequent processing should, and should not, be applied to data items and datasets.

B. Analytic Webs

The totality of data produced and consumed by a working scientific team, combined with all the processes used to transform and analyze those data, comprise a scientific data processing enterprise, that can be formally represented by what we call an *analytic web* [6], [12], [22]. An analytic web should provide facilities for producing and accessing all such data as well as its associated metadata. We propose that analytic webs be realized by two types of closely interrelated graph structures, namely *dataset derivation graphs* (DDGs) and *process derivation graphs* (PDGs). The purpose of a DDG is to organize datasets into a structure based upon the way in which the datasets are derived from each other. The purpose of a PDG is to define precisely the processes by which these derivations are performed. Moreover, execution of a PDG results in the automatic creation of a DDG for each input dataset upon which the PDG is executed.

In this paper, we provide a concrete example of an analytic web, using the Little-JIL process definition language [8], [29], [30], to define a PDG and to produce the DDG. We use this example to illustrate why rich semantic language features are needed to support scientific workflow.

II. Motivating Example

Measuring and forecasting water flux and storage in the ecosystem is of tremendous importance to society, and a central focus of major sci-

entific investigation efforts, such as NEON (http://www.neoninc.org/) and the Waters Network (http://www.watersnet.org/). Such forecasts require detailed hydrological measurements of natural and human-dominated ecosystems. These measurements come from networks of real-time sensors and are subjected to elaborate real-time adjustments and to considerable, perhaps iterative, postprocessing over ensuing months or years.

A group of ecologists at the Harvard Forest Long-Term Ecological Research site (http://harvardforest.fas.harvard.edu/) is designing a real-time system to calculate change in water storage using the water balance equation: $dS = P - ET - Q$, where equation inputs come initially from five real-time data streams.

- **Precipitation** $(\boldsymbol{P})$: 15-min precipitation totals $(P1, P2)$ measured at two rain gauges to guard against missing data due to sensor drift and failure.
- **Surface Discharge** $(\boldsymbol{Q})$: 15-min average stream flow values measured at a stream gauge.
- **Evapotranspiration** (ET): 30-min average *ET* values measured at an eddy-flux tower.
- **Photosynthetically active radiation** (PAR): 30-min average *PAR* values which can be used to estimate *ET* when it cannot be accurately measured directly

This system will incorporate three features that are typical of virtually any sensor network and raise challenging issues for dataset development and provenance documentation.

1) **Real-time quality control** entails nontrivial processing, much of it determined on the fly, which may cause different data items in a dataset to have different process provenance. For example, such systems may incorporate duplicate sensors (here, $P1$ and $P2$), and elaborate rules for value selection. Thus, in this case, two precipitation measurements are taken, with rules about what to do when one sensor fails to deliver a value, and different actions to be taken if the values differ by different specified amounts.
2) **Regularly scheduled postprocessing of data** (e.g., after 30 days) is used so that individual imputed data values can be computed using models that take into account values coming from both preceding and subsequent measurements.
3) **Alternative past measurements** also may become available and may then be included in additional datasets. This may be desirable, for example, to make use of recent measurements that did not arrive in time for real-time processing, to correct earlier measurements as a result of later detection of sensor drift, or to replace or impute faulty or missing values with measurements from other sources.

Our experience showed that these features significantly complicated attempts to model the Water Flux data processing process and that powerful process modeling language features helped to address these complications.

III. Proposed Model of an Analytic Web

In this section, we describe formalisms used to define the graphs comprising analytic webs.

A. Dataset Derivation Graph

A DDG documents the specific data items or dataset instances created when a dataset developer applies processes, using specific tools and subprocesses on specific input data items or dataset instances. A DDG thus is a representation of the *process provenance metadata* needed to specify precisely how a dataset was generated and to support reproduction of the dataset by other scientists. The DDG illustrated here uses rectangles to represent specific data items and dataset instances and ovals to represent specific tools or subprocess instances. There is an edge from each data item and dataset instance node to the process instance node from which it was derived. Each oval process instance node is connected by one or more edges to the data item(s) and dataset instance(s) that it used as input(s) to derive the indicated output data item or dataset instance. Each time a process is executed, a new set of data items and dataset instances is created, and these data items and dataset instances, as well as the process instances that created them, must be added as nodes of the evolving DDG. Each DDG node instance can be stored independently with a unique URL for identification.

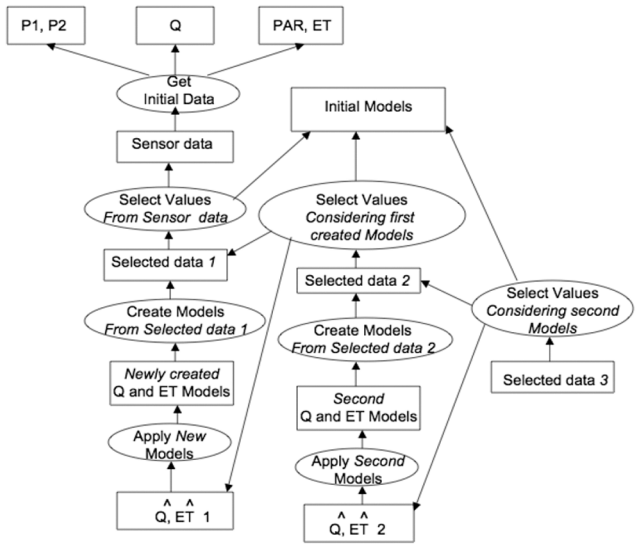

Fig. 1. DDG showing the complete provenance of Selected data 3.

To make this clearer, Fig. 1 illustrates a DDG that could have been generated by executing a sequence of tools and subprocesses. This DDG shows the result of two iterations through a processing loop, resulting in the creation of three instances of Selected data, denoted here by Selected data 1, Selected data 2, and Selected data 3. Selected data 1 is simply the set of data items that resulted from filtering initial real-time data using specific filtering models. This generally results in a dataset where some data item values are missing, most often due to intentional deletion. Selected data 2 results from applying to those dataset models that have been created as a consequence of examining Selected data 1, thereby filling in missing data item values and replacing others. Selected data 3 results from applying subsequently created models to Selected data 2. Although not shown, each of the data items and tools represented in this figure would also contain metadata annotations with detailed information such as the version of the tool, execution platform, parameter settings, scientists involved, and date of creation or derivation. This DDG provides documentation of the exact processing steps that were taken to produce these datasets.

B. Process Derivation Graphs

A PDG is a precise representation of the procedural steps that might be used to process data items and datasets in a scientific process. Many different formalisms could be used to define PDGs. But our example suggests that semantic issues, such as concurrency, abstraction, exception handling, and agent specification, are important to the clear specification of actual scientific processes. In this paper, we use Little-JIL, a visual process definition language, originally developed for defining software engineering processes, to define PDGs.

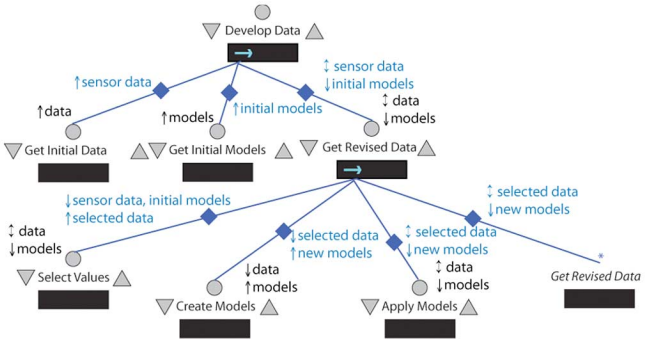

Fig. 2. Little-JIL PDG whose execution could generate the DDG in Fig. 1.

C. Little-JIL as a PDG Definition Language

A process is defined in Little-JIL using hierarchically decomposable steps [29], [30] where a step represents a task to be done by an assigned agent. A step, moreover, is much like a procedure, in that it is defined once in a process, but can then be invoked from other points in the process simply by reference to that step. This facility encourages the creation of modules and their reuse and also supports iteration through the recursive invocation of a step from within the step itself. Fig. 2 shows a Little-JIL definition of a PDG whose execution could generate the DDG shown in Fig. 1. We use this example to illustrate some key features of Little-JIL.

A Little-JIL step is represented by a solid black rectangle on top of which is a small disc, annotated by a specification of the types of the step's parameters. Thus, the step Get Revised Data requires two arguments, the first is used both as input and output, and is of type data, and the second is used only as input and is of type models. Each nonroot step is connected to its parent by an edge annotated with the arguments passed between parent and child. Thus, the step Get Revised Data receives from its parent the arguments Sensor data (generated by Get Initial data and used as input and output) and Initial models (generated by Get Initial Models and used as input). The right arrow in Get Revised Data's step bar indicates that its four substeps are executed in left to right order. Note that Get Revised Data is called recursively, and the presence of a Kleene star on the edge to the recursive call creates an iteration that produces the successive instances of Selected data shown in Fig. 1. On exiting the recursion, the final value of Selected data becomes the new value of Sensor data.

Comparison of Figs. 1 and 2 indicates the need for both the DDG and PDG graph types. A PDG shows how artifacts can be created, but a DDG documents the precise way in which specific artifact instances have been created by a particular process execution. Thus, for example, Fig. 2 defines specific step sequences and a specific recursion. However, it does not indicate how many times the recursion will continue. Indeed, until more details of the leaf steps are elaborated (not done here due to lack of space) it is impossible to know how a recursion is terminated. The DDG in Fig. 1, however, is a precise documentation of the two iterations, leading to the generation of the three successive instances of Selected data.

Little-JIL incorporates a number of other language features many of which are particularly effective in supporting the definition of scientific processes. Those used in subsequent examples in this paper are described here briefly.

1) **Resources and Agents***:* Each Little-JIL step specifies the type of agent responsible for the step's execution. Little-JIL agents may be either humans or automated devices and, in some cases, either might be appropriate.

2) **Substep Cardinality***:* An edge between a Little-JIL parent and substep may be annotated by a cardinality specification defining the number of times the substep is instantiated. This may be a fixed number, a Kleene * (for zero or more times), a Kleene + (for one or more times), or a Boolean expression.

3) **Step Sequencing***:* Every nonleaf step has a *sequencing badge* (an icon in the left portion of the step bar; e.g., the right arrows in Fig. 2), which defines the order in which its substeps execute. A sequential step (right arrow) indicates that substeps are executed sequentially from left to right. A parallel step (equal sign) indicates that substeps can be executed in any (possibly arbitrarily interleaved) order. A choice step (circle slashed with a horizontal line) indicates that the agent executing the step chooses among any of the step's substeps. A try step (right arrow with an X on its tail) mandates a sequence in which substeps are to be tried until one is successful.

4) **Channels***:* *Channels* are used to synchronize steps and to deliver artifacts produced by identified source step(s) to identified destination step(s).

5) **Requisites***:* *Requisites* are optional steps that enable the checking of a specified condition, either as a precondition for step execution or as a postcondition to assure that step execution has been completed acceptably. A requisite it represented iconically an arrowhead either to the left or the right of the step bar.

6) **Exception Handling***:* A Little-JIL step can define *exceptional conditions* when some aspects of the step's execution fail (e.g., one of the step's requisites is violated). This violation triggers execution of a matching exception handler associated with the parent of the step that throws the exception. *Exception handlers* are steps attached by a red edge to a red X on the right side of a parent step bar. They define how thrown exceptions are to be handled. An edge connecting an exception handler to its parent is annotated with the type of the exception being handled and an indication of how execution is to continue after the exception has been handled.

7) **Scoping and Recursion***:* The parent step and all of its descendants represent a *scope*, specifying what data are considered local to that scope. This is particularly useful in defining the context for recursive execution of a Little-JIL step.

IV. Using Little-JIL to Define and Execute a PDG and to Create a DDG

We now use Little-JIL to define the Water Budget process precisely and to illustrate semantic features that are useful in addressing some of the difficult features of this process. Most parameter and artifact annotations are omitted from these examples to make them easier to read and save space.

A. The Sensor Data Management Step: Use of Decomposition and Concurrency

The step Sensor Data Management (Fig. 3) is at the heart of the Water Budget process. It is a parallel step decomposed into three substeps: Get Measurements, which collects data from the sensors in real time; Process Data, which preprocesses the data; and Model Stream Data Gen, which applies models to the real time data to build datasets. Because the step is a parallel step, each of its substeps is able to execute asynchronously, as the problem domain demands. The collected data and created models are communicated between substeps, both as parameters and via channels that are declared in Sensor Data Management and are thus accessible to all of its substeps. Here, we describe a representative set of steps, their Little-JIL definitions, and the DDGs that could be generated by their execution.

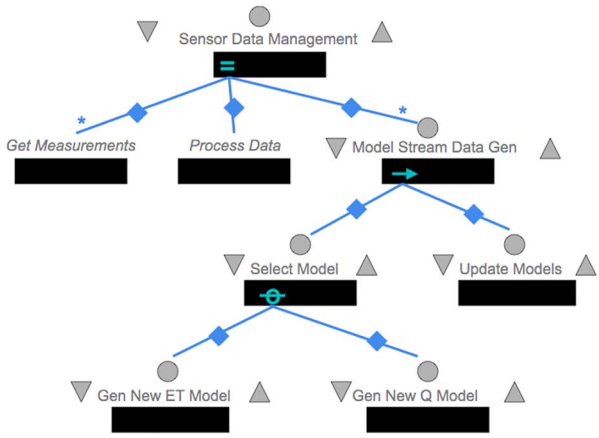

Fig. 3. Sensor Data Management step.

B. The Get Measurements Step: Support for Multiple Data Streams

The Get Measurements step (Fig. 4) reads and processes the values from the sensors and sends the results to the real time input stream. This is done by having the different subprocesses, Get Met Station Data, Get Flux Tower Data, and Get Stream Gauge Data, take responsibility for examining data from three different sources. Each of these substeps can execute independently and in parallel with the other two, and each may throw a different type of exception if difficulties arise with their sensors.

The Get Met Station Data step has two substeps, Get P1 and Get P2, each of which deals with one of the two precipitation gauges. The substeps that acquire the individual data items are responsible for annotating each data item with some provenance information. At present this consists of rudimentary metadata, specifically a date/time stamp and a quality flag.

C. Handle MS Sensor Timeout Step: Use of Exception Management

Each Get measurements substep acquires data from its sensors and also defines what is to be done if exceptional conditions arise. The Get Met Station Data step deals with situations where some, or all, of the expected data items do not arrive by throwing an exception. For example, the substep Get P1 attempts to obtain a reading from meteorological sensor 1. If this access succeeds, then the value is passed as P1 annotated with the observation date and time and with the *measured* quality attribute. The Get P1 step is also responsible, however, for determining when new P1 data is not available and then, subsequently, for throwing an exception that is to be handled by Handle MS Sensor Timeout step, passing the identifier of this sensor (namely sensor 1) as a parameter. The other two substeps of the Get Measurements step, Get Flux Tower Data and Get Stream Gauge Data, carry out their responsibilities similarly to Get Met Station Data.

The Handle MS Sensor Timeout step (Fig. 5) is the handler for the Sensor Timeout exception that can be thrown in the Get Met Station Data step. It begins by choosing, based on the value of the input parameter, Reread Precipitation 1 or Reread Precipitation 2. If this succeeds, then the output of this step is annotated with the date and time and the "measured" quality flag. If the step fails (e.g., because the sensor is inoperative), a Sensor Down exception is thrown and the Sensor Down handler executes the Get Airport step to obtain the reading from a local airport. If the Get Airport step also fails, it throws an exception, caught by the Put Null Reading step of the Get Airport exception handler, to produce a null value for P1 (or P2) and a quality flag "missing."

D. Using the Process Definition to Create the DDG

The above process results in the creation of two data items, P1 and P2 each of which might have been derived in a number of different ways. Specifically (using P1 as the specific example), we have the following.

1. P1 arrives in a timely fashion and is recorded.
2. P1 does not arrive in time, a timeout exception is thrown, Reread Precipitation is executed, and P1 is obtained.
3. P1 does not arrive in time, a timeout exception is thrown, Reread Precipitation is executed, P1 still does not arrive, a Sensor Down exception is thrown, the Airport Read step is executed, and P1 arrives.
4. P1 does not arrive in time, a timeout exception is thrown, Reread Precipitation is executed, P1 still does not arrive, a Sensor Down exception is thrown, the Airport Read step is executed, P1 does not arrive, an Airport Data Read Failure exception is thrown, the Put Null Value step is executed, providing a null value for P1

The differences among these four possibilities are important and thus the current process attaches to P1 a quality flag having one of the following values:

missing—no measured value is available;
estimated—a measured value is available from another location (e.g., the airport);
measured—the measured value is available.

Using this process, P1 is annotated as *measured* in the first two cases, (making it impossible to distinguish between them), as *estimated* in the third case (but without documenting the measurement location), and as *missing* in the fourth case, again leaving out the details describing what alternatives had been tried. We note, however, that each of the four different sequences of process steps can be thought of as a different trace through the process, illustrating the importance of annotating each value with process provenance information, as provided in the DDG.

Examples of the DDGs that represent cases 3 and 4 are shown in Fig. 6. Note that the boxes in this figure represent actual data instances, namely the actual data values that are bound at execution time to the type specifications in the process definition. Thus, for example, one of the boxes at the top of Fig. 6 is annotated by sensor1 null @ "try1 time" indicating that this box represents the actual (null) value that was delivered at the specific time, "try1 time."

The ovals in Fig. 6 represent the process step instances that were the actual producers and consumers of the actual data items. Thus, there is an oval labeled Get Airport that indicates that an instance of the Get Airport step was used to generate the data item in the box shown below this oval. This step instance represents the instance of Get Airport that was invoked as the process's response to the two null readings. Two arrows from this oval connect it to two boxes, representing the fact that the values represented by these boxes were used as inputs to the step represented by that oval. In this case, the use that is made of these data items is simply to note that they are both null, causing the Get Airport step to be executed to produce this output. Other ovals may make more substantive use of their inputs in generating their outputs. Thus, for example, in the left-hand DDG of Fig. 6 the result of the execution of the instance of the Get Airport step is an actual value, annotated with date and time information, which is taken as the final value of P1. In this case, no actual step is used to generate that value, and instead the DDG indicates that the value is produced as a consequence of the parameter binding operation that occurs as an integral part of the execution of every step. The fact that this oval does not represent an actual step is indicated by the use of italics in its annotation. In the other case, a null reading is obtained, and a null value is then the final value of P1. Thus, such DDGs provide more useful

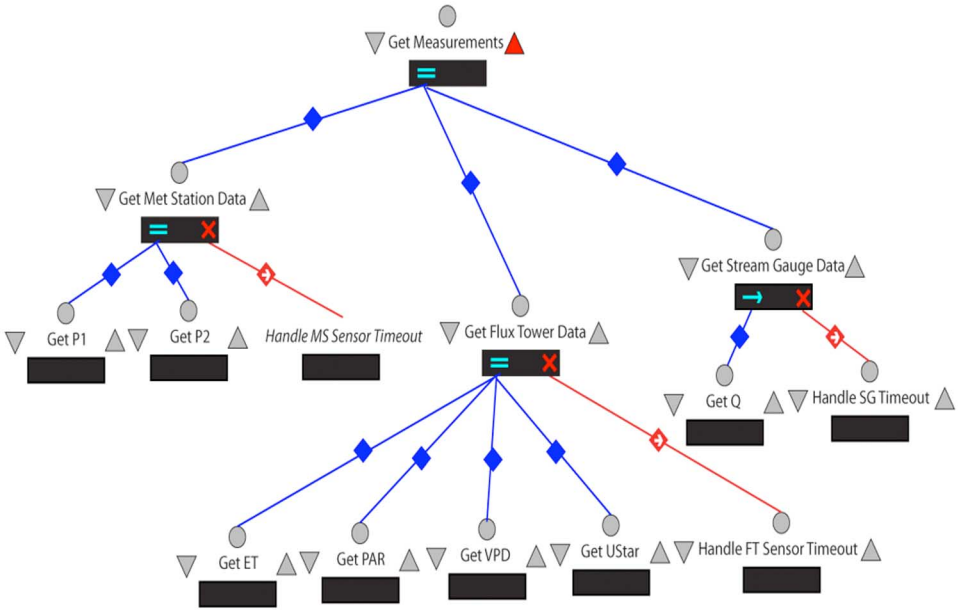

Fig. 4. Get Measurements step.

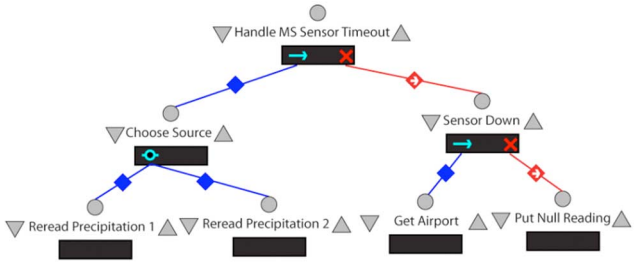

Fig. 5. Handle MS Sensor Timeout exception handler step.

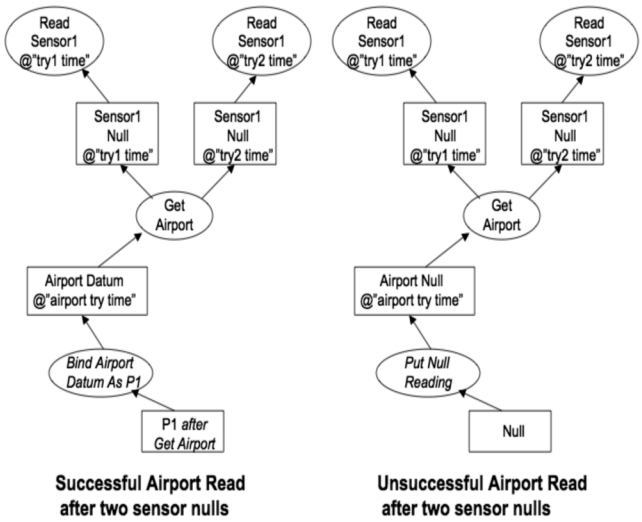

Fig. 6. DDG of the original acquisition of P1.

information about the provenance of the resulting value of P1 than a mere annotation, *measured, estimated*, or *missing*.

V. Evaluation

The example processes in Section IV have illustrated how an analytic web provides provenance information and highlights some semantic capabilities, such as hierarchical decomposition, procedural invocation, concurrency, and exception management that are needed to define modern scientific processes. There is a striking similarity between these needs of the scientific community and what is generally provided by modern programming languages. This supports our intuition that scientific processes bear some strong similarities to computer software, and thus the challenges of defining them have strong parallels with the challenges of programming complex software systems. Scientific processes, however, also require specifications that are easily understood by scientists, which suggests the value of a visual representation. Thus, it is not surprising to find that a visual process language that incorporates the salient control features of a modern programming language can be an effective tool for defining scientific processes.

The example processes in Section IV also show how DDGs can be built incrementally as the execution traces of a scientific process, captured as directed acyclic graphs. Since these DDGs increase in depth as process execution progresses, the DDGs derived from executing lengthy processes can become large and cumbersome. However, although the pictorial depiction of an entire DDG is large, its internal representation is a typical tree-like structure that should be amenable to terse internal representation. In addition, different, terse views of the full DDG could be provided in response to focused queries.

VI. Related Work

There are numerous other scientific workflow projects, many of which have been presented at meetings, such as [9], [25], and [28]. Most of these projects (e.g., Kepler [2], [3], [19], Taverna [21], [31], and JOpera [17], [23]) base their specification of process flow upon the use of various kinds of data flow graphs (DFGs). For example, Kepler is based upon Ptolemy II [5], [11], [24], which uses a powerful and flexible DFG structure to specify how datasets can be moved between processing capabilities. Kepler integrates a broad range of support tools that help with such key activities as specification, execution, and visualization of scientific data processes. Chimera [10], [15], [16] was one of the earliest scientific workflow systems. It emphasized

the use of pictorial visualizations to represent scientific processes. Chimera's pictorial representations were also in the form of a DFG. Taverna [21], [31] is a more recent system that seems to focus on supporting the integration of web services, particularly for the creation of bioinformatics applications. Taverna's integration mechanism is a workflow notation that is also based upon a DFG formalism. More recently, JOpera [17], [23] has suggested the use of XML to specify scientific workflows as plug-ins that could be integrated using Eclipse. JOpera workflows also are based upon the use of a DFG formalism to represent scientific processes. Teuta [13], [14] represents scientific processes through UML diagrams that offer some features, such as limited forms of concurrency, that go beyond the semantic features of a basic DFG. We believe that the Water Budget process described here illustrates aspects of scientific processes that cannot be easily captured using DFGs. Other UML diagrams can be effective in capturing some of these features, such as agent specification (using "swim lanes"), and some forms of concurrency, message passing, and exception handling. Too often, however, this requires the use of different UML diagrams, which seems less desirable than Little-JIL's ability to capture all of this using a single diagram.

Much work is also aimed at documenting the provenance of scientific datasets. Many of the approaches to provenance documentation are summarized in [26] and [27]. Indeed, these approaches have been compared to each other more formally in [20]. These approaches seem to fall generally into two categories. In one approach (e.g., [1], [7], and [18]), each data artifact generated by execution of a scientific process is stored in a database, and the artifact's provenance can then be obtained by recursively querying the database. The second approach entails building a derivation graph on the fly as execution of the scientific process proceeds (e.g., Kepler [4] and [15]). Our own approach falls into this latter category. We note that these two approaches are essentially equivalent to each other. Both collect provenance information by documenting the execution trace that has led to the creation of the data artifact being documented. In the former, the provenance structure is created on demand, and in the latter, the derivation structure is built incrementally. Again, the Water Budget examples illustrate how a more precise process definition can lead to a very precise model of data provenance, as illustrated with the DDGs generated here.

VII. Future Work

We believe that the semantic features in Little-JIL present a useful starting point for considering the features that should be incorporated into languages for defining the PDG. Further investigation of the essential requirements for the semantics of a PDG is needed. Specific details of DDGs also require further evaluation. For example, we need to evaluate various internal representations of DDGs to determine how to store them efficiently while still supporting efficient creation of needed visual representations. Moreover, datasets represented by the nodes of the DDG could be regenerated from scratch or cached to expedite generation of subsequent datasets. Specific strategies for determining when and what to cache should be the subject of future research. While DDGs can be used as the basis for the creation and attachment of process metadata to datasets, further research is needed to determine how this is done best.

The value of an analytic web will be greatly enhanced by the availability of tool sets that support such capabilities as the creation of the PDG, the execution of the PDG, the automatic creation of the DDG, viewing and querying these graph structures, and reasoning about the soundness of the scientific processes defined.

Considerably more value would be obtained by creating a facility for supporting the automation of some or all of the parts of a process defined by a PDG. Indeed, because Little-JIL is an executable process language, PDGs defined in Little-JIL can then be executed by Juliette, the Little-JIL interpreter. A crude early prototype system, SciWalker was used to show that this is possible. SciWalker delivered datasets to tools, human participants, and subsystems as prescribed by the PDG, and also picked up outputs produced by participants to move them forward to next processing steps. SciWalker, however, did not produce DDGs. We now have plans to produce SciWalker 2, which would carry the SciWalker capabilities much farther. Specific goals of SciWalker 2 include automatic generation of DDGs from the executions of a Little-JIL PDG, optimizers to reduce the size of DDGs, viewers to help users understand Little-JIL PDGs and DDGs, analyzers to support the need for users to understand their Little-JIL PDGs and to identify process defects, and visual editors to support both building new Little-JIL PDGs and modifying them. Indeed, our view of SciWalker 2 is that it should be essentially an environment that integrates diverse tools in support of the definition, execution, and analysis of scientific processes defined in Little-JIL.

Finally, we believe that the best way to make progress in developing the ideas just outlined is to continue to create analytic webs to represent scientific processes of various kinds. Our work with ecological processes is encouraging, yet preliminary. We hope that there will be much more work of this sort, not just in ecology, but also in the representation of processes in a wide range of other sciences.

Acknowledgment

The authors are grateful to many colleagues who supported this work and contributed key ideas. In particular, they wish to thank E. Riseman, A. Hanson, D. Jensen, D. Foster, J. Hadley, P. Kuzeja, H. Schultz, B. Rawert, G. Avrunin, and M. Raunak for their advice, support, encouragement, and many stimulating conversations.

References

[1] A. Aiken, J. Chen, M. Stonebraker, and A. Woodruff, "Tioga-2: A direct manipulation database visualization environment," in *Proc. Int. Conf. Data Eng.*, 1996, pp. 208–217.

[2] I. Altintas, C. Berkley, E. Jaeger, M. J. B. Ludäscher, and S. Mock, "System demonstration, Kepler: An extensible system for design and execution of scientific workflows," in *Proc. 16th Int. Conf. Scientific Stat. Database Manage.*, 2004, pp. 423–424.

[3] I. Altintas, A. Birnbaum, K. Baldridge, W. Sudholt, M. Miller, C. Amoreira, Y. Potier, and B. Ludäscher, "A framework for the design and reuse of grid workflows," in *Proc. Int. Workshop on Scientific Appl. Grid Comput.*, 2005, pp. 119–132, series Lecture Notes in Computer Science, nr 3, Springer-Verlag GmbH, 2005. ISBN 3-540-25810-8.

[4] I. Altintas, O. Barney, and E. Jaeger-Frank, "Provenance collection support in the Kepler scientific workflow system," in *Proc. Int. Provenance and Annotation Workshop, Provenance and Annotation of Data*, 2006, vol. 4145, pp. 118–132.

[5] P. Baldwin, S. Kohli, E. A. Lee, X. Liu, and Y. Zhao, "Modeling of Sensor Nets in Ptolemy II: Information Processing in Sensor Networks," in *Proc. Inf. Process. Sensor Netw. (ISPN)*, Apr. 26–27, 2004, pp. 359–368.

[6] E. R. Boose, A. M. Ellison, L. J. Osterweil, L. A. Clarke, R. Podorozhny, J. L. Hadley, A. Wise, and D. R. Foster, "Ensuring reliable datasets for environmental models and forecasts," *Ecol. Inform.*, vol. 2, no. 3, pp. 237–247, 2007.

[7] P. Buneman, S. Khanna, and W. C. Tan, J. Van den Bussche and V. Vianu, Eds., "Why and where: A characterization of data provenance," in *Proc. Int. Conf. Database Theory*, 2001, pp. 316–330.

[8] A. G. Cass, B. S. Lerner, E. K. McCall, L. J. Osterweil, S. M. Sutton, Jr, and A. Wise, "Little-JIL/Juliette: A process definition language and interpreter," in *Proc. 22nd Int. Conf. Softw. Eng.*, Limerick, Ireland, Jun. 2000, pp. 754–757.

[9] B. F. Cooper and R. S. Barga, in *Proc. IEEE Int. Workshop on Workflow and Data Flow for Scientific Applications SIGMOD Record*, 2006, vol. 35, pp. 54–56.

[10] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman, J. Nabrzyski, J. Schopf, and J. Weglarz, Eds., "Workflow management in GriPhyN," in *Proc. 14th Int. Conf. Scientific Stat. Database Manage.*, 2003.

[11] S. A. Edwards and E. A. Lee, "The semantics and execution of a synchronous block-diagram language," *Sci. Comput. Progr.*, vol. 48, no. 1, pp. 21–42, Jul. 2003.

[12] A. M. Ellison, L. J. Osterweil, J. L. Hadley, A. Wise, E. Boose, L. A. Clarke, D. R. Foster, A. Hanson, D. Jensen, P. Kuzeja, E. Riseman, and H. Schultz, "Analytic webs support the synthesis of ecological data sets," *Ecology*, vol. 87, no. 6, pp. 1345–1358, Jun. 2006.

[13] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto, and H. Truong, "ASKALON: A tool set for cluster and grid computing," *Concurr. Comput.: Pract. Exp.*, vol. 17, no. 2–4, pp. 143–169, 2005.

[14] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, and M. Wieczorek, "ASKALON: A grid application development and computing environment," in *Proc. 6th IEEE/ACM Int. Workshop Grid Comput.*, Nov. 2005, pp. 122–131.

[15] I. T. Foster, J.-S. Voeckler, M. Wilde, and Y. Zhao, "Chimera: A virtual data system for representing, querying, and automating data derivation," in *Proc. 14th Int. Conf. Scientific Stat. Database Manage.*, 2002, pp. 37–46.

[16] I. T. Foster, J.-S. Vöckler, M. Wilde, and Y. Zhao, "The virtual data grid: A new model and architecture for data-intensive collaboration," in *Proc. Conf. Innovative Data Systems Research*, 2003, p. 11.

[17] T. Heinis, C. Pautasso, and G. Alonso, "Mirroring resources or mapping requests: Implementing WS-RF for grid workflows," in *Proc. 6th IEEE Int. Symp. Cluster Comput. Grid (CCGrid2006)*, Singapore, May 2006, pp. 497–504.

[18] D. P. Lanter, "Design of a lineage-based meta-data base for GIS," *Cartogr. Geogr. Inf. Syst.*, vol. 18, no. 4, pp. 255–261, 1991.

[19] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the Kepler system," *Concurr. Comput.: Pract. Exp.*, vol. 18, no. 10, pp. 1039–1065, 2006.

[20] L. Moreau *et al.*, "The first provenance challenge," in *Concurrency and Computation: Practice and Experience*. New York: Wiley, 2007.

[21] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li, "Taverna: A tool for the composition and enactment of bioinformatics workflows," *Bioinform. J.*, vol. 20, no. 17, pp. 3045–3054, 2004.

[22] L. J. Osterweil, A. Wise, L. A. Clarke, A. M. Ellison, J. L. Hadley, E. Boose, and D. R. Foster, "Process technology to facilitate the conduct of science," in *Proc. Software Process Workshop, Lecture Notes in Computer Science*, 2005, vol. 3840, pp. 403–415.

[23] C. Pautasso and G. Alonso, "The JOpera visual composition language," *J. Vis. Lang. Comput.*, vol. 16, pp. 119–152, 2005.

[24] [Online]. Available: http://ptolemy.eecs.berkeley.edu/ptolemyII/

[25] P. Herrero, M. S. Pérez, and V. Robles, Eds., in *Proc. 1st Int. Workshop Scientific Applications of Grid Comput., SAG 2004*, Beijing, China, Sep. 20–24, 2004, Revised Selected and Invited Papers. Lecture Notes in Computer Science 3458 Springer 2005, ISBN 3-540-25810-8.

[26] Y. L. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance techniques," Comput. Sci. Dept., Indiana Univ., Bloomington, IN, Tech. Rep. 612, 2005.

[27] Y. L. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-Science," *SIGMOD Rec.*, vol. 34, no. 3, pp. 31–36, 2005.

[28] Proc. 16th Int. Conf. Scientific and Statistical Database Management (SSDBM 2004), Jun. 21–23, 2004. Santorini Island, Greece, IEEE Computer Society, ISBN 0-7695-2146-0.

[29] A. Wise, A. G. Cass, B. S. Lerner, E. K. McCall, L. J. Osterweil, and S. M. Sutton, Jr, "Using little-JIL to coordinate agents in software engineering," in *Proc. Autom. Softw. Eng. Conf.*, 2000, pp. 155–163.

[30] A. Wise, Little-JIL 1.5 Language Rep., Dept. Comput. Sci., Univ. Massachusetts, Amherst, MA, UM-CS-2006-51, 2006.

[31] K. Wolstencroft, T. Oinn, C. Goble, J. Ferris, C. Wroe, P. Lord, K. Glover, and R. Stevens, "Panoply of utilities in Taverna," in *Proc. 1st IEEE Int. Conf. E-Science and Grid Technol. (E-Science)* , 2005, pp. 156–162.