A Software Engineering Approach to Scientific Data Management
Corietta L. Teshera-Sterne, May 2010

1. Introduction

1.1 A changing science

An ecologist stands under an enormous hemlock tree deep in the New England forest. She marks down the number of a tag on the tree on a sheet of paper, measures its girth, and notes its health and what plants are growing around it. After examining several dozen more trees, she will put her notes in a filing cabinet with similar ones from past years, and eventually the ecologist or one of her students will analyze the collected data and publish their conclusions.

A few yards from the hemlock stands an 80-foot tall scaffolding structure, steadied by guy wires. The sensors this eddy-flux tower holds above the forest canopy take measurements of environmental variables such as atmospheric turbulence, carbon dioxide flux, photosynthetically active radiation, and wind speed every few milliseconds, year-round, then wirelessly send them to a database accessible by anyone with an Internet connection. Over the next few years, teams of researchers from several universities across the country will download the data, integrate it with measurements from stations located in other ecosystems, perform analyses, and potentially publish many papers on different aspects of the collected data.

The ecologist represents a traditional kind of science that, while essential, is rapidly being eclipsed by the advancement of technology. New data collection capabilities, such as sensor networks, combined with powerful computational techniques make it possible for new, large-scale questions to be investigated in disciplines such as particle physics, systems biology, climate change, and ecosystem ecology. At the same time, the scale of the datasets involved creates equally large data management challenges. Many of these issues rest simply on the fact that it can be difficult to keep track of what has been done with any subset or to the whole of any given scientific dataset. A single dataset used in the new information-intensive sciences is no longer a physical folder of plot inventory sheets or even digital pages in a spreadsheet, but often is a complex synthetic product of several initial sets. Data frequently undergoes quality control and other preliminary computational steps before analysis is even begun. A lack of knowledge about these procedures can further complicate data management and analysis in a variety of ways. Metadata about the origins and subsequent transformations of data is called *provenance*, and its production and use is an important challenge facing today's scientific community.

The term "provenance" itself is defined as origin or source, often with the more specific connotation of the history of a physical object. In the scientific context, taxonomic studies are informed by knowledge of the origin of individual specimens. In paleontological research, for example, much of the value of a fossilized specimen is provided by its accompanying provenance information. "Origin" provenance, for a fossil, may include information about where and when it was found and by whom [1].

An additional perspective is gained by looking at what can be termed "derivation" provenance: the record of what has happened to the specimen since discovery, such as taxonomic reclassifications, transfers of ownership, and any damage, preparation, or other modifications. This collected knowledge makes up a large part of the specimen's value, providing the possibility of recovering additional specimens or making comparisons between specimens from different

locations, as well as verifying scientific significance by identifying fakes or illegal trafficking. In the case of fossils, a lack of such provenance information may render the specimen essentially unusable as part of a scientific collection (although it may retain value in a non-research educational context) [1-3]. In a broader application to scientific research, the value of origin and derivation provenance applies not only to physical specimens, but any object of inquiry, including pure data.

1.2 Why provenance?

One of the major reasons provenance information is essential to scientific data management, made clear by the hemlock research scenario at the beginning of this paper, is simply that data collection and processing methods do change over time. Whether due to advancements in methods or simply disciplinary and researcher preference, such changes may introduce inhomogeneities that can lead to incorrect analyses and incompatibilities between data processed at different times. Thus, the technological advances that make large-scale data collection possible are themselves reflected in the final data products of some long-term datasets.

One example of a biased data product resulting from a change in data processing comes from the domain of climate science, as reported by Karl *et al.* [4]. Climatologists are interested in tracking the area of polar snow cover as an indirect measurement of changes in global temperatures and hydrological patterns, which may in turn help predict the effects of future changes in environmental conditions on the global climate system. Satellite imagery has provided the means to conduct measurement of snow cover in vast, previously inaccessible polar regions. Data-reduction algorithms are used to produce a time series of annual average snow cover, relying on monthly averages of snow cover for each region. In 1981, a change in the primary method for calculating these monthly averages introduced a bias into the data, resulting in erroneous increases in estimated snow cover. The results were subsequently published in influential international climate change policy recommendations such as the reports of the Intergovernmental Panel on Climate Change. Karl *et al.* noted that problems due to such changes in processing algorithms would be likely only to increase as climate science relied more on such methods, and as traditional manual measurement systems were replaced with electronic systems. Therefore, they advocated for the parallel development of a standard methodology to track these changes. With the institutionalization of provenance capture, the consequences of such changes in data-processing can be made transparent, and older data can be more easily re-processed with new methods to produce homogeneous datasets. Nearly two decades later, the same issues are only compounded with the complexity of current scientific sensor systems. The same argument can be applied to the operational problems inevitably encountered with the sensors themselves. Outright sensor failure may be relatively easy to detect, but more subtle malfunctioning such as a slow drift in accuracy may be more difficult. In that case, provenance data may be essential on several fronts: it allows for identification of affected data values that then may be adjusted or excluded, and frequent review of the intermediate data values may allow for detection of sensor drift itself.

Another key issue is that as data collection and analysis become more complex, they also tend to become distributed. The scientists responsible for producing the satellite snow cover data, for example, were from different institutions than the meteorologists who subsequently used that

data to make conclusions about the snow cover's extent. Thus, a divide can be created between data producers and data consumers. Imagine that the forest ecologist referred to earlier moved from studying hemlocks to a new project, and a colleague inherited her filing cabinet of data and notes along with the results of some initial quality control and statistical tests. The colleague would use the notes (and perhaps correspondence with the original researcher) to decipher what had already been done with the data in order to accurately continue the analysis. She would then also be able to decide what steps to repeat if she found errors or disagreed with the original procedures, and to reduce her workload by identifying subsets that might not need revision. The same type of transfer occurs in information-intensive large-scale science, but is likely to be on a more indirect basis. Data producers, such as the scientists in charge of the eddy flux tower, are more frequently making their data accessible on the Internet to facilitate collaborative research. Funding agencies, too, are moving toward requiring publicly released data to encourage transparency. The consumer, therefore, can potentially access large datasets without much contact with the producers, and without full knowledge of how the data were produced. Access to provenance information can alleviate concerns about the reliability of conclusions based on such shared data.

One aspect of data reliability, and a major goal of increasing access to data, is the reproducibility of experiments. This is one of the essential components of good science: a hypothesis is only truly considered to be supported by the results of an experiment if it can be shown that an independent reproduction of the experiment will produce the same results. From the point of view of the traditional scientist, it may be tempting to think that the widespread use of computers to automate scientific data-management tasks will make reproducibility easier because, at least theoretically, the same computation performed on the same data inputs should execute identically every time. However, this is highly dependent on how well those computational operations and inputs are recorded, and provenance capture is currently far from standard in scientific computation. Also, in addition to the previously discussed issues of a producer-consumer gap, computer-based data manipulations are far from unified. Software used for scientific computing ranges from ad-hoc programs written by individual researchers to spreadsheets, statistical packages, large proprietary or open-source software suites, and web-based services. Additionally, many different techniques may be used for a single dataset. Coordinating the use of multiple software products increases the difficulty of capturing provenance. There are also issues of how best to store and access metadata, especially from multiple sources. Producing provenance information and storing it in a useful way, therefore, is a valid and pressing concern from the perspective of both experimental science and the scientific software that supports it.

Other types of synthetic information products have similar issues with reliability and user confidence, and have developed various solutions that are illustrative. Distributed and community-based online projects involving many participants, in particular, face issues of user confidence. The Distributed Proofreaders component of Project Gutenberg [5] is a web-based effort to digitize and provide open access to literary works in the public domain. Each original paper volume undergoes a digitization and quality-control process analogous to the production of scientific data sets. Optical character recognition (OCR) software provides an automated method for digitizing texts, but the reliability of the program's output is limited by the clarity of the

original print. The OCR output is corrected by a series of volunteer proofreaders ranked by experience, with more highly ranked participants checking the work of newer ones, and is divided into sections so multiple teams can work on the text at once. The entire volume is then reassembled and uploaded to an Internet archive. The results of scholarly analysis of historically important literary works may depend on the accurate reproduction of exact text, so the participant responsible for each activity is recorded throughout this process to ensure the ability to trace issues. It is also simple to then compare the final data product (the proofread text) to the original (the scanned text page), although this option is neither available to nor required by the vast majority of users. Another project where such comparisons are integrated directly into the data product is the online encyclopedia Wikipedia, which has both the strength and weakness of granting universal permission to edit any content on the website. A "History" tab allows readers (that is, data consumers) to view all previous changes made to the information, making it easy to identify malicious or erroneous additions and, just as importantly, "roll back" to an earlier edit. Online efforts like these illustrate an additional facet of provenance capture: in either case, the information would be unlikely to be recorded if it was left to the participants or if it interfered in any way with the process. To be truly useful and comprehensive, the recording of this information must be both automatic and unobtrusive.

1.3 Defining and capturing provenance

A first step in capturing useful provenance information is to rigorously define it in terms of particular types of information that can be captured from the execution of scientific data processes. As the term is used here, a scientific data process is a specified sequence of steps or tasks that either takes in data inputs or produces them, performs manipulations or other actions, and subsequently produces some form of data product. Manipulations performed on data may occur at the level of the dataset (merging sets from more than one source, for example) or individual data value (such as taking a value as input to calculate a subsequently derived value). The most basic form of output would be a simple transformation of the input; however, as analysis and data visualization software become ever more sophisticated, this output might extend to complex data products such as rendered and possibly interactive animations or other visualizations. What is considered to be a complete process is largely a matter of scale: on one extreme, it could include the entire experimental scientific procedure itself, from hypothesis to a conclusion based on collected and analyzed data. Most approaches to defining and automating such processes, however, focus on either the most computationally intensive or operationally complex segments of the procedure. Here, we are more concerned with processes involving some subset of actions performed on the data itself; however, I believe that it is important to keep in mind that data management and analysis generally do occur within the context of larger scientific inquiries.

The most straightforward form of a specification for such a scientific process would be a simple, sequential "laundry list" of tasks. Provenance, in that case, would be a corresponding list of what tasks were actually performed: each time an action was executed an entry would be made recording the fact. This could perhaps be extended with a record of intermediate data products and descriptions of any problems encountered in following the list. But this approach provides little support for describing the complexity of real scientific processes, which may

include branching decision trees, parallel execution of sub-processes, and other nonlinear components. The motivating example for this research has been the implementation of a hydrological sensor network at the Harvard Forest ecological research site in Petersham, Massachusetts, with the goal of producing a detailed water budget of small forested watersheds. This network includes several types of instrumentation such as stream flow and precipitation sensors, and the aforementioned eddy-flux tower. Each type of sensor produces data that must undergo a variety of quality control and preliminary processing procedures. These procedures must be able to handle a variety of situations where data is missing, derived from multiple sources, or have unexpected values. They also often include complex handling of multiple sub-processes that themselves must include such data handling. The dataset is ultimately made publicly available on the internet [Fig. 1].

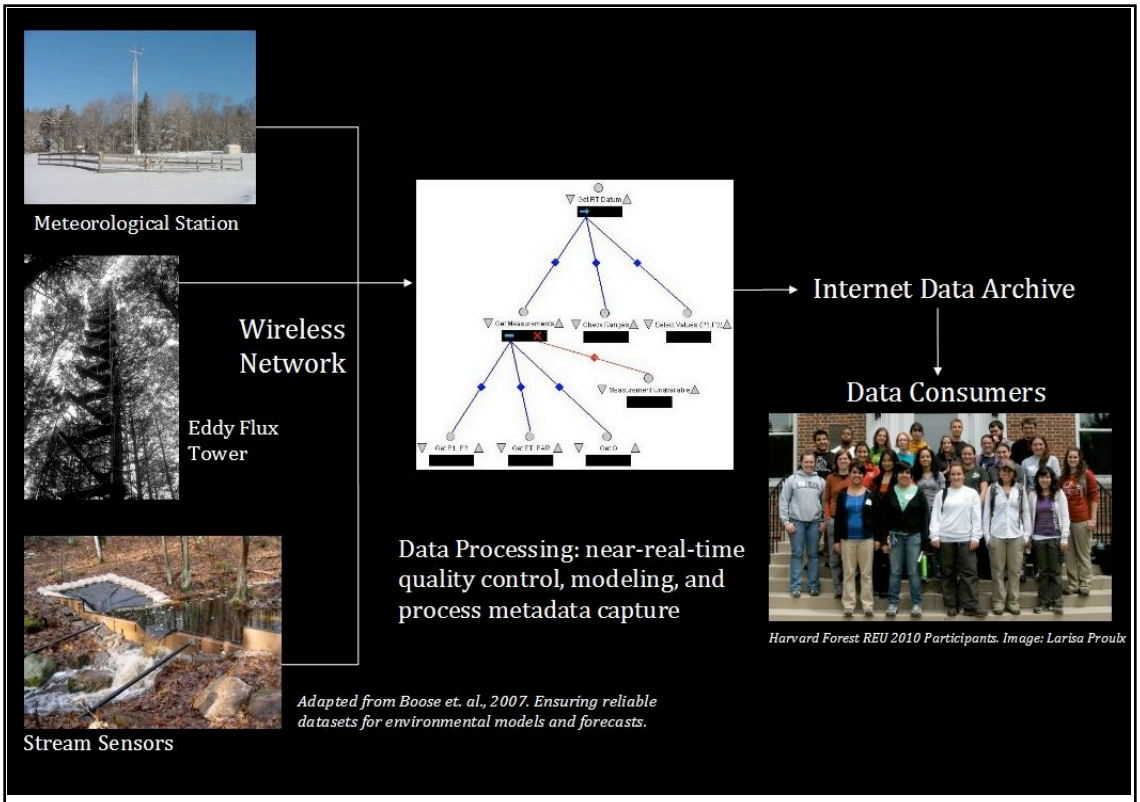

Figure 1. *Harvard Forest Hydrological Sensor Network*
Data is collected from three types of hydrological sensors, processed, and made available to data consumers.

The methods used for capturing provenance, then, must also be able to handle such complexity. In addition, even simple, linear processes raises questions of exactly what metadata is important to record, which could include information such as timestamps, metadata from software and external computational services, and researcher annotations. A more complete concept of provenance would provide further information about why certain paths were taken and the resulting relationships between intermediate data products and computational actions. The title of this research project is based on the idea that the execution of scientific processes

such as hydrological data processing, and their associated provenance, can be rigorously defined using software engineering concepts.

2. Related works

2.1 Workflows

A number of surveys exist that provide an overview of the many approaches to capturing and accessing provenance that have been investigated [6-10]; here I will outline a subset of approaches to introduce some of the issues encountered in the course of this research project. In recent years, interest in provenance in scientific computation has resulted in a research community approaching the issue from a variety of perspectives, such as those addressing domain-specific problems or particular parts of data management and analysis.

Attendant to the advantages of advancing scientific technology is an increase in the introduction of errors in the sequence of events and details followed in such tasks. This problem only increases when automated or semi-automated tasks are performed by a network of multiple researchers, computer services, and other agents. To address this issue, a number of projects have developed software for the creation and execution of scientific workflows. At the simplest level, a workflow may simply be the aforementioned "to-do list" of tasks to be performed, but many scientific workflow software tools provide the ability to execute computational tasks involving complex controls of data. Many of the designers of such software also recognize the importance of providing additional tools to capture data provenance information as the workflows are executed. These projects provide insight to some of the inherent issues encountered in capturing and using provenance information.

One such workflow program is the Kepler system, which demonstrates the dataflow-based, visual representation of computational tasks employed by many workflow systems [Fig. 2] [11]. The addition of data provenance capture to Kepler has been approached in several ways. The framework described by Altintas *et al*. [12] emphasizes the potential to not only capture provenance such as data inputs and intermediate values, but to track the evolution of workflows themselves as a tool for debugging and improvements in workflow design. Additionally, when a rerun of the workflow is required due to changed input parameters, it provides a basis for using analysis of provenance information to save execution time by only rerunning required portions of the workflow. Another project, pPod, is based on the specific needs of the biological systematics

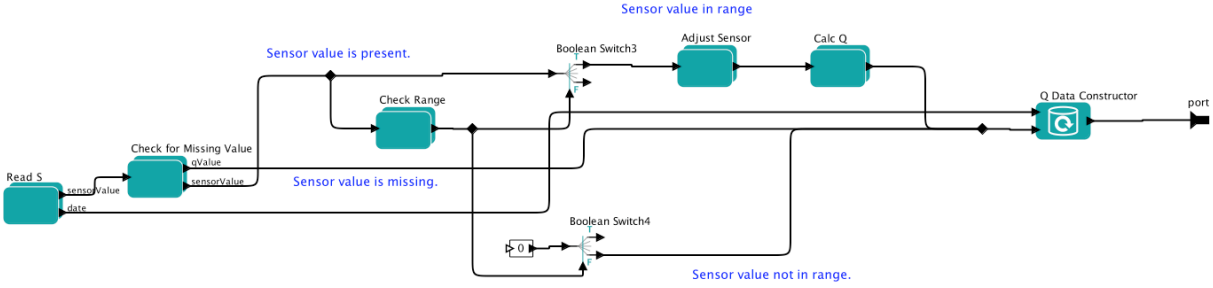

Figure 2. *Kepler Workflow Diagram*

This diagram illustrates the workflow approach to defining scientific processes.

community using Kepler to support a large-scale project to reconstruct organismal phylogenies [13]. pPod captures provenance data through the addition of tokens to a stream, where each token represents a provenance-related event produced by an actor. The system then uses the collection of tokens to construct provenance graphs representing the relationships between actors and the dependencies between data items. It is a relatively common approach to store data provenance information in a directed acyclic graph (DAG) representing the relationships between some combination of information about and dependencies within dataflow, often with additional information about the computational entities that use and transform data.

These and the similar dependency graphs constructed by many other provenance-inclusive workflow systems often make some attempt to include relevant information from each computational step beyond input and output objects. An issue that then arises is the problem of capturing information from "black boxes" that might not provide sufficient information about themselves. The ability to take advantage of remote computational services becomes more important as in-silico experiments increase in complexity. At the same time, service-based processing presents problems such as, for example, an inability to query the service for relevant information like the current state. A model developed for a different workflow system, Taverna, approaches this by attaching annotations to provenance captured from whole saved workflow instances [14].

Another issue inherent in science involving large quantities of data is how to visualize and understand large-scale trends contained in the data. It can be difficult if not impossible to derive overall meaning by simply viewing large spreadsheets of data; instead, scientists view data in ways that range from simple graphs to complex rendered animations and other interactive, computationally intensive visualizations. The workflow system VisTrails [7, 15, 16] is designed specifically to handle data integration for visualization. It is novel for being one of the relatively few workflow systems to integrate provenance support as part of its initial design, as well as for the level of abstraction provided both in the workflow and the provenance data.

The first International Provenance and Annotation Workshop in 2006 (IPAW '06) provided an opportunity for communication among researchers from diverse projects, including many of those mentioned here. Out of this meeting, the First Provenance Challenge was developed to provide the community with an opportunity to compare the systems used and the associated approaches to provenance. The participating teams demonstrated their ability to run (or simulate) a common workflow and capture some level of provenance information [17, 18].

Although much development has gone into workflow systems such as those discussed here, these programs for the most part approach provenance through the development of extensions and not as a core design feature. More generally, it has become readily apparent that the provenance community lacks formal representations for the definition of dataset production and processing, especially in terms of handling complexity. In many cases, the data-flow based workflow diagrams are not sufficient for reproducibility and other concerns of data producers, and provenance trails pose difficulties for (potentially different) data consumers.

A different and more robust way of providing a formal representation of scientific processes has been developed by an interdisciplinary team of computer scientists and ecologists, drawing both on software engineering concepts and previous ecological metadata-definition techniques. The analytic web concept brings together a workflow-style data-flow diagram with

formal definitions of the potential process (a process definition graph, or PDG) and the process as actually executed (data-derivation graph, or DDG).

3. A software engineering approach: process programming and the Analytic Web

3.1 Process programming

Although scientific workflows appear to be a fairly comprehensive solution to scientific process execution, they have a number of characteristics that tend to present challenges to provenance capture. However, workflows are not the only conceptual approach to the specification and automation of scientific tasks. A different, innovative concept is to define and describe a process using software engineering ideas.

The idea of *process programming* was introduced more than two decades ago in the domain of software engineering by Lee Osterweil, who in the title of a paper presented at the 9th International Conference on Software Engineering in 1987 [19] declared that "Software processes are software too". That is, the processes that are performed during the development and construction of a software product can themselves be described using software engineering concepts. This kind of meta-description allows engineers to improve software development by creating robust, formal definitions of the procedures they follow, which both decreases the probability of procedural errors and provides greater access to overall process knowledge for individual workers. What distinguishes this concept from workflows most, however, is that this kind of thinking confers the ability to take advantage of software engineering concepts that provide a more robust semantic environment for defining processes.

Although software engineering may not intuitively appear to be connected to the management of scientific datasets, there are two extensions to this concept that make it more broadly applicable. First, Osterweil and his colleagues envisioned the development of a compilable programming language that would facilitate the writing and execution of process programs. Second, it was recognized that this concept translates to a more general idea of process beyond the immediate domain of those who developed the idea of process programming. If the development of software products can be described through such methods, so can the similar development of other products, and ultimately, any analogous process that consists of a structured collection of tasks. The hydrological data network provides one example of the utility of this thinking in the form of exception handling. Many conventional programming languages support a mechanism by which the majority of cases of, for example, input data are processed in the normal course of the program. A case that would, for example, invalidate assumptions made in the rest of the program, however, is passed to a handler that can safely take care of the exceptional case before returning control to the program. This provides a more robust solution than a simple "if-then" statement as is available in most workflow diagrams, as well as explicitly distinguishing between normal and exceptional cases. An example would be assigning a negative value to a variable that is expected to be positive, such as hydrological information about streams that are physically unable to contain negative amounts of water. A negative value, in this case, would need to be substituted in some way in order to be processed, and may indicate a sensor problem that would trigger some kind of corrective action.

Coupled with the development of an executable programming language in which to write process programs, this idea gains applicability to a wide range of domains that could benefit

from rigorous definition of tasks. It is intuitive that such a definition could then lead to the rigorous capture and storage of provenance information produced by such tasks.

3.2 Little-JIL

A current implementation of this concept is in the form of a language called Little-JiIL, developed at the LASER lab at the University of Massachusetts, Amherst [20]. A full description of the language is available elsewhere [21], but a short overview is a necessary reference for further discussion of this project. Little-JIL is best described as a *coordination language*, visualizing processes in terms of items of work called *steps* that are assigned to entities that are capable of doing work, known as *agents*. The capabilities of agents can range from a "dummy agent" that simply starts and completes steps, to the more autonomous and complex decision-making abilities of a piece of software or a human researcher. A step is represented by an icon that displays its name and other information such as control flow, and are connected into a graph that represents the entire process. Steps may represent and control a single action, or may be composed of a hierarchical series of sub-steps that must be completed before the ancestral step completes. A step in which the assigned agent "does work" directly without further control input from the process is called a *leaf* step and has no children. Most kinds of steps take in input parameters and produce output in the form of objects called *artifacts* that can be passed throughout a process. Processes, as composed of steps and the connections between them, are created and edited in a graphical environment [Fig. 3].

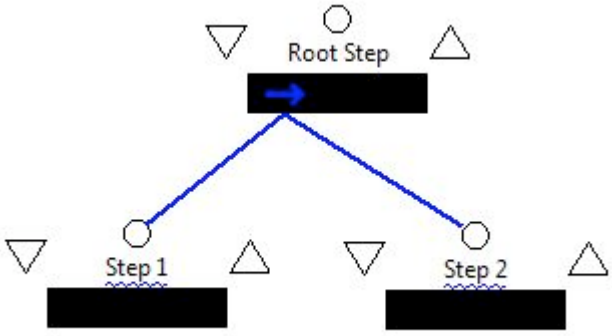

Figure 3. *A simple Little-JIL process*

This process consists of three steps: a root step with two leaf steps as children. The arrow badge on the root step indicates that the children execute sequentially from left to right.

In the context of scientific data management, Little-JIL provides the ability to coordinate the activities of agents used in these kinds of processes, such as multiple computer programs, remote data sources, and off-line actions or decisions by human researchers. And as discussed, one of the main strengths of Little-JIL in the scientific context is the ability to apply programming concepts such as exception handlers and iterative loops to the processing of scientific data.

3.3 The Analytic Web

While the process programming approach, with the use of Little-JIL, can provide a more rigorous process definition than that achieved through a workflow diagram, more documentation is required to fully describe an executed process [22-24]. Briefly defined, an Analytic Web is a collection of three graphs that together describe a process and its execution [25]. Like process

programming itself, the Analytic Web concept was originally developed for the definition of software engineering processes. In the context of this research, the concept has evolved to serve both the needs of the ecologists with whom this project has been developed in collaboration, and the primary goal of investigating provenance capture.

i. DFG

The first graph of an Analytic Web, the Data Flow graph (DFG), represents the process at a high level in the same form as the data-flow diagrams central to many scientific workflow programs. This has the advantage of being intuitive to scientific users who either use workflow programs or are otherwise familiar with defining processes in similar forms, and provides a clear visual representation of the process. However, the DFG is less semantically powerful than the PDG, as it lacks coordination features (for example, exception handling) that provide precise definitions of the process [26]. It is possible that including the DFG to some degree in the current incarnation of the Analytic Web may prove useful in proposed future projects to translate scientific tasks into Little-JIL, or to integrate Little-JIL with established workflow tools.

ii. PDG

As has been discussed, the usefulness of provenance information is correlated with the precision with which the process itself is defined. This process definition can in of itself be thought of as useful provenance information in the form of *prospective provenance*. In other words, the definition of all possible all execution paths restricts the captured provenance to a subset of a larger known set of possible paths. In the Analytic Web concept, this is represented by a Process Definition Graph (PDG), which corresponds to the Little-JIL process diagram.

iii. DDG

The natural counterpart to the prospective provenance encoded in a PDG is *retrospective* provenance, the execution trace of the process as it actually occurred. This is represented by the third component of an Analytic Web, the Data (or Dataset) Derivation Graph (DDG). A DDG stores provenance information in a directed acyclic graph (DAG) consisting of nodes containing the intermediate data values (Data Instance Nodes, or DINs) and the computational elements that produce them (Step Instance Nodes, or SINs), and arrows that represent derivation relationships between them (it should be noted that the current convention is for the arrows to point *up* from a node to its parent, reflecting the retrospective nature of the trace). The term "derivation" as it used here deserves defining, as the derivation of one value from another forms a central part of this conceptualization of provenance. As defined by Merriam-Webster, "to derive" is to "take, receive, or obtain, esp. from a specified source". This is trivially true, but lacks any deeper explanation of the possibly manipulative relationship between the source and the object being received. The dictionary definition of "derivation" is somewhat more useful, evoking a logical connotation as a "sequence of statements showing that a result is a necessary consequence of previously accepted statements". In the context of data objects, however, what does it mean to be a "necessary consequence"? It may be of use to consider a simple process in which an initial data value (data object A) is passed as input to some computational step (step 1), which returns a new value as output (data object B). Therefore,

- *B is derived from A.*

There are at least two ways to think about what actually happened during the process:

- *A triggered the step that produced B* (which can be taken as either "the presence of A triggered the execution of step 1, which produced B" or "the creation of A by a prior step triggered the execution of step 1, which produced B").

- *the production of B is the result of a manipulation performed on A*. Or, A is modified in a specified way (by the step) to produce B.

Taken together, these views indicate that this modification plus the original value, within the defined structure of the process, creates the *derived* value. This relationship is denoted in the DDG as an arrow pointing from B up to the source of the modification (the step), and from there to A. If this seems to be a too-detailed a discussion of an apparently simple concept, it is because exploring the concept to this level is necessary for resolving questions of how to fit different elements into a DDG. This definition also has a couple of other important consequences. First, both pieces of this relationship (the modification and the value) can theoretically be quantified or otherwise defined, stored, and reexamined. When scaled to the level of a dataset, this has the desired result of a complete trace of the process as executed. Second, when the derivation relationship is defined in this way, it reinforces reproducibility and opens the possibility of eliding certain parts of the execution trace, because the relevant pieces can be "re-derived". This may be helpful for storage or visualization issues.

It should be noted that the modification may be the same as no action if the output value is, for whatever reason, equivalent to the input. However, this still represents a successfully executed computational element with a result (or, in Little-JIL terms, a successfully completed step), and therefore the definition of derivation still holds for this case. This will be significant for future development of precisely what elements should be included in a DDG. Further discussion of the current state of the Analytic Web, and of DDGs in particular, requires a more complete example introducing the complexity of real-world processes.

4.0 Motivating example and experience in creating a DDG

4.1 Description of study system

As previously mentioned, the motivating example for this research is a proposed hydrological sensor network at Harvard Forest, which involves tracking the inputs, outputs, and internal activity of water in a small forested watershed to gain insight into how variations in environmental conditions affect hydrology. Producing a detailed understanding of water budgets has previously been made difficult because of the scale of measurements needed to account for subtle sources of water flux, especially atmosphere-forest water exchange (evapotranspiration), which is now made possible by the eddy covariance method. The physical network consists of an array of scientific instrumentation measuring a range of hydrological variables. Data is communicated to an ecologist's computer and undergoes quality control and other initial processing. Final datasets are uploaded to Harvard Forest's online data archive [27]. The initial network is composed of three major sensor installations: a meteorological station measures precipitation (*P*), an instrumentation tower takes canopy-level evapotranspiration measurements (*ET*), and a series of gages measure stream discharge (*Q*). These environmental variables are related with the equation $P - ET - Q = dS$, where dS is a measurement of the total change in

water stored in watershed components such as groundwater and vegetation.

The current research was most closely focused on using data processing from the stream-flow sensors as an initial example process. The data is derived from weirs installed on Big and Little Nelson Brooks**,** the parallel outlet streams of the Black Gum Swamp wetland (a second set of gages in the form of pipes (culverts) are installed on another stream, Bigelow Brook, above and below a beaver swamp. The pipes and weirs operate on the same principles, but installation decisions depend on the hydrological and physical characteristics of the stream banks.) The weirs consist of a reinforced dam with a rectangular spillway and a V-shaped notch of a specified size and shape. A sensor located in the catchment area behind the weir measures water pressure, which can then be converted to flow with a linear equation relating pressure to area using the known area of the notch. At the time of this research, the data was stored in an onsite data logger and downloaded to a portable device at predetermined intervals, although a local wireless network was under construction to allow near-real-time data streaming from the data logger for processing and uploading to the internet.

Processing consists of several sequential steps for quality control and transformation of initial readings into stream discharge measurements. As discussed earlier, one of the issues associated with this system in particular, and such networks in general, is detection of and adjustment for sensor malfunction. The data is range-checked to provide a broad determination of sensor function at the level of individual values, then adjusted by a specified equation for known sensor drift. The final stream-flow data product is then calculated from this adjusted value. A simple data flow diagram of this process is shown in Fig. 4.

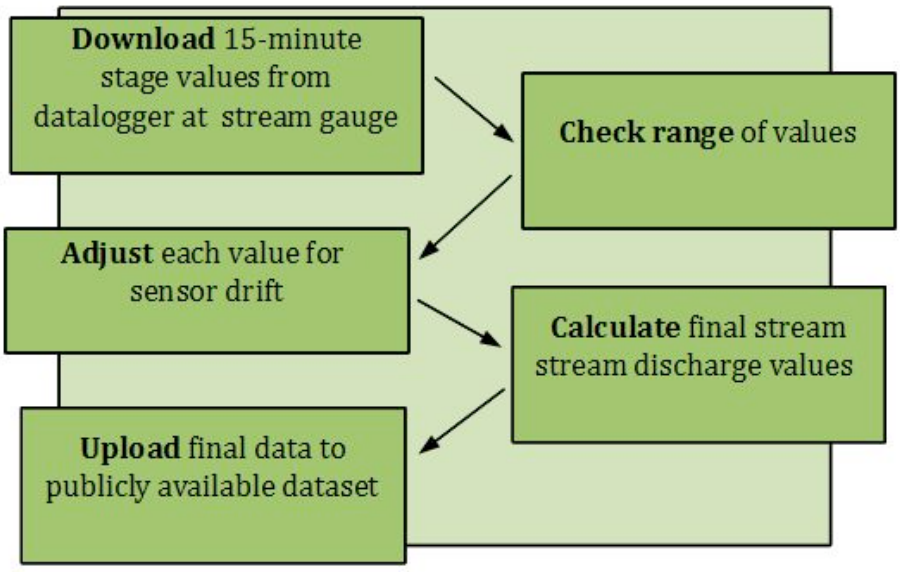

Figure 4. *A data-flow representation of the stream process*

This graph represents the process as designed and followed by the ecologists responsible for the processing of hydrological data.

Some (hopefully large) percentage of data values will undergo this process as described, but there are a number of ways in which errors can occur and be corrected. Errors such as

missing data may be the result of intermittent equipment or transmission errors, or ongoing situations such as winter ice buildup. These errors may be corrected by various gap-filling or modeling approaches, such as substituting an average of several previous data values. The process for such corrections could easily be more complicated than the original; for simplicity, we filled in a value of zero for such errors. However errors are handled, it is important that it does not interfere with the subsequent processing of the remaining data.

A second sub-process of the larger water budget system is the quality control and processing for the precipitation sensors. The data processing is similar to that of the stream sensors, except that two sensors take measurements in parallel. During processing, data is preferentially derived from the primary sensor unless errors are detected, in which case data is read from the second sensor dataset. If both sensors return unreliable data, the program accesses a remote service to obtain data from a nearby airport (a missing or gap-filled value is substituted if the airport is unable to provide a usable value). This decision makes the process both more complicated from a process programming perspective and more interesting from a provenance perspective.

4.2 Translation into an Analytic Web

The data-management procedure for the Harvard Forest stream sensors was an appropriate first candidate for translation to an executable Little-JIl process for several reasons. The ecologists collaborating on the project were interested in an automated software solution as processing was done manually using spreadsheet software, which was error-prone and required a researcher time commitment. In addition, the process is fairly simple in that it is both short and essentially linear in the most common case, but still has the potential to exhibit interesting behavior depending on the results of computational elements such as the range check. The construction of this process illuminated some key challenges in the development of DDGs that capture complete and useful provenance information [Fig. 5]. The precipitation process adds the complexity of parallelism, and served to reproduce the procedure developed for creating a running Little-JIL process and DDG [Fig. 6].

The Little-JIL diagram represents at once the description of the process, the software for coordinating its execution, and the PDG for provenance purposes. As such, there is a combination of elements included in the program for computational purposes (that is, the process as designed by the ecologists), those included to take advantage of a programming approach and facilitate the transition to and satisfy the constraints of the Little-Jil language, and those included to aid provenance capture.

In addition to the use of exception handling as described earlier, the stream process PDG takes advantage of other Little-JIL features. One such element was the decision to make the Check Range step a post-requisite to the Read S step. That is, the step that controls the reading in of a sensor value (S) will not complete, and control will not pass to the next step (Adjust S), until the Check Range step has successfully executed. If Check Range fails, an exception handler interrupts the process to perform specified actions to correct the error, and the process resumes on the next data value. This behavior is one of the features of Little-Jil that makes the language appropriate for this type of process automation. However, this configuration is not easily drawn from the initial data flow diagram, which essentially treats each computational step as

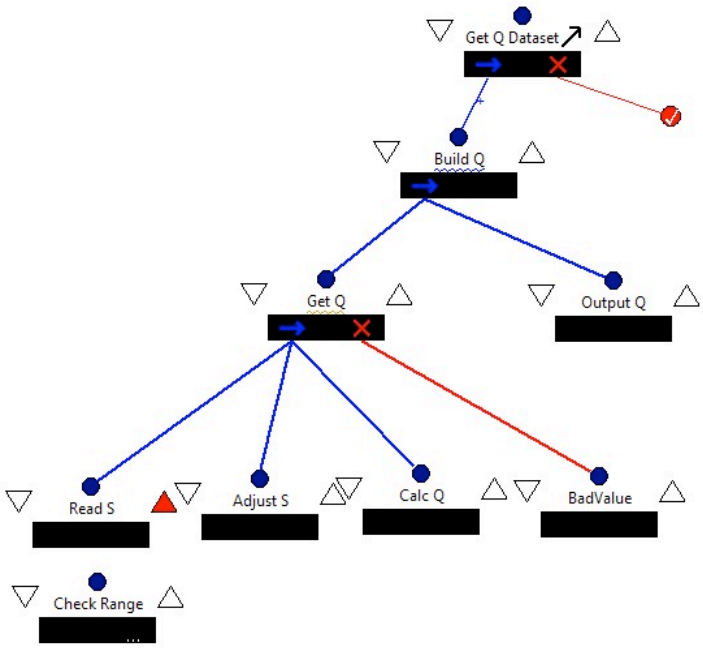

Figure 5. *Stream flow PDG*

The stream flow process as encoded in Little-JIL. Note the exception handlers (red) and Check Range as a requisite for Read S.

equivalent. The difference here is that all other steps explicitly change the data object; for example, the Calc Q step takes in an adjusted sensor output data value (in the form of a SensorData object), performs a calculation, and produces a final discharge value (a QData object). The Check Range step, in contrast, can be thought of as *not* having an "output" in the same sense. Rather, the result of the evaluation of the passed-in value is an action: either the range check is passed and the process can be continued, or the check fails and an exception handler is triggered.

There are several possible ways of representing this in the DDG produced by a process with this kind of step. One would be to simply not include the Check Range step because it does not fit into the "value plus modification" view of derivation. It was generally agreed, however, that being able to answer the question of whether the range check actually occurred is important to the ecologists who designed the process. Another solution, then, would be to create a third type of node, or include it as a special kind of annotation. Annotations are by definition extra information added to the graph, rather than an integral component; additional types of nodes add complexity and should not be considered lightly. The eventual result of this discussion was to include Check Range as a normal SIN, with the next step being linked both to Check Range and to the parent step that produces the output value.

This discussion also led to the definition of three rules of DDG structure, extending the previous definition of a DDG as a DAG composed of Step Instance Nodes and Data Instance Nodes. These rules serve both as a guideline for the development of processes and as a check for the correct creation of DDGs from a process:

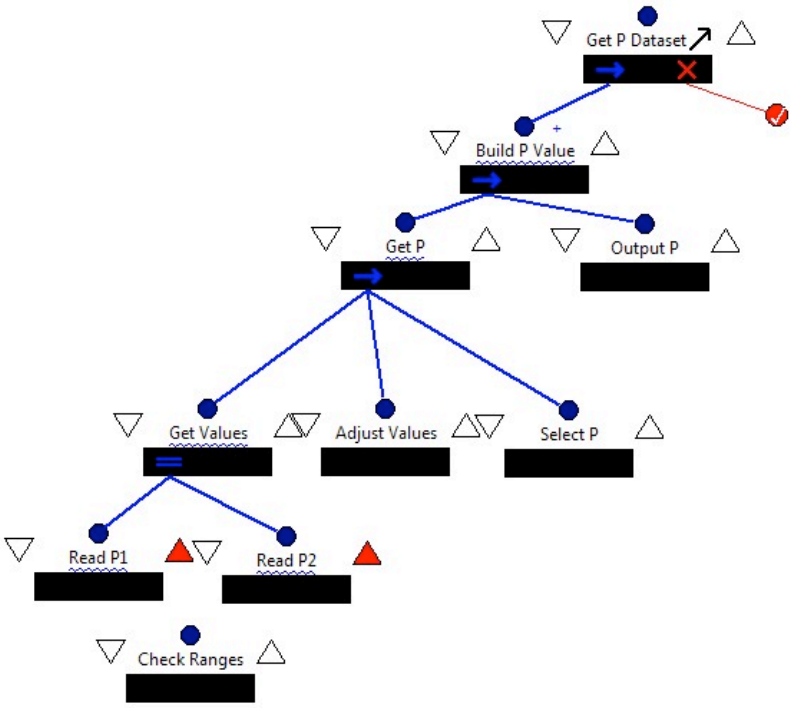

Figure 6. *Precipitation process PDG*

The precipitation process as encoded in Little-JIL. Note the parallelism at the step "Get Values".

1) A DDG always starts with a SIN.

This rule is somewhat for the sake of consistency between DDGs, and to avoid chicken-and-egg arguments of whether a process begins with input data or with an action performed on that data. I argue that a process may be defined as beginning with the *production* of data itself, and therefore have no inputs; furthermore, even a process with inputs must begin by somehow reading them into the process.

2) DINs must be preceded and succeeded by SINs (DINs can only link to SINs)

Because arrows in a DDG represent derivation, derivation is the result of a modification on a value, and modifications can only be performed by steps, one data object in a process cannot be derived from another without the intervening execution of a step.

3) SINS can be preceded and/or succeeded by either SINs or DINs

This rule is most informed by the Check Range situation, where the completion of Read S and the beginning of the next step (or alternatively, the triggering of the exception handler) occurs without modification of the value passed to Check Range. This rule, therefore, also reinforces the solution to the Check Range question proposed above. A process could, in theory, consist only of steps such as Check Range, with the flow of control being modified by the value being passed to a step, but the value itself not being produced as output.

5. Java program

A second part of this project was the development of another form of DDG production

mechanism, and a departure from the Analytic Web concept: a recreation of the process as a program written directly in Java rather than Little-JIL. One of the justifications for a Java implementation of the program was to provide a way to refocus on the development of the DDG. The Java program executes exactly the same procedure as the Little-Jil stream process, and in a way represents the status of much small-scale scientific computing. As mentioned before, the stream sensor data was, at the time of this research, processed by data transformations in a spreadsheet. This kind of special-purpose data processing program is frequently written by researchers as an ad-hoc solution for the automation of tedious tasks, intended to address immediate concerns of decreasing error rates and the time needed for processing rather than larger-scale concerns of reproducibility and provenance. The program developed here, however, demonstrates that provenance can be captured and stored in a useful form such as a DDG even from these kinds of programs.

The program reuses many of the same components from the Little-JIl process, which was aided by the fact that in the Juliette environment, Little-JIl agents are written in Java. Exception handling, for example, is an essential part of Java programming and therefore exception classes were transferred between the programs unchanged. The data parsing, input/output, and methods for actual data processing such as sensor adjustment were also copied verbatim. In the absence of Little-Jil's diagrammatic approach to control flow, a single driver class iterates through each data value read into the program and calls the appropriate data processing methods. Like the original, the Java program captures process metadata at the time of creation or modification of data objects, and at the time of execution for the method calls that replaces step instances. Nodes are created and linked to the DDG by a single "addMethod" or "addData" method call for adding computational elements (in the form of strings representing called method names, analogous to step instances) or data objects, respectively. The potential pitfalls of this type of program mostly lie in the fact that, although they are designed to be as minimally intrusive to the program as possible, the inclusion of the method calls to capture and store DDG elements are not syntactically enforced. This violates the two principles discussed at the beginning of this paper, that in order to be effective, provenance capture should be both automatic and unobtrusive. Additionally, the DDG-creation code has various levels of domain-specificity, with the result that extending this program to other processes may require a certain amount of retrofitting. Aside from these issues, the program produces both valid data output and a DDG that follows the previously discussed format. The program would also be fairly easily extended to implement the precipitation process as well.

6. Conclusion

There is much further development needed on this project, especially in the details of the construction and storage of DDGs. Development of the storage of DDG information has, until now, taken a secondary position to practical concerns of how to capture data items and theoretical concerns of what data to include and leave out of the DDG. The current format is to output the information stored in a DDG as a hierarchical XML document. However, this has always been intended as a temporary measure to facilitate the production of easily visualizable experimental DDGs, not as a permanent storage solution. One issue is that because the DDG may include many intermediate values for every final value, and may include an arbitrary

amount of information collected from steps, it has the potential of being many times the size of the initial or final datasets. This raises not only questions of storage but of whether it is practical to avoid storing re-derivable parts of the DDG at all. It also may make sense to develop what goes into a DDG and how it's stored concurrently, because the one necessarily influences the other. Visualization is a related concern, especially whether abstraction is practical to hide unnecessary (or sensitive) details from some viewers while displaying more relevant information.

Addressing these concerns is increasingly important because data management will only get more complex as science continues to become more data-intensive. For example, the National Science Foundation (NSF) recently announced new data sharing requirements for research funded by the agency. Instead of simply requiring that data be made publicly available at some point, the new policies require a comprehensive, peer-reviewed data-management plan to be submitted with grant applications and research proposals [28]. Increasing collaboration between scientists is one of the stated goals of such policies; however, in order to be shared effectively researchers need data to be accompanied by compatible information about the data's origins.

On a personal level, my experience talking to researchers across many scientific domains and in fields such as engineering and medicine has left me convinced that nearly every member of a data-intensive discipline has experienced some instance in which easily available provenance information would improve their ability to use data. The development of provenance capture mechanisms such as DDGs, therefore, will have far-reaching effects in today's data-filled world.

7.0 References

1. Palmer, C.P, Boyd, D.W., and Yochelson, E.L. 2004. The Wyoming Jurassic fossil *Dentalium subquadratum* Meek, 1860 is not a scaphopod but a serpulid worm tube. Rocky Mountain Geology 39, 2, 85-91.

2. Montero, A and Dieguez, C. Types of Paleontological Collections, Interest: The Case of the Museo Nacional De Sciencias Naturales (MNCN), Madrid, Spain. *In International Symposium and First World Congress on Preservation and Conservation of Natural History Collections, Vol 2*. 221-227.

3. Moreau, L., Groth, P., Miles, S., Vazquez-Saldeca, J., Ibbotson, J., Jiang, S., Munroe, S., Rana, O., Schreiber, A., Tan, V. and Varga, L. 2008. The Provenance of Electronic Data. *Communications of the ACM* 51, 52-58.

4. Karl, T.R., Quayle, R.G. and Groisman, P.Y. 1991. Detecting Climate Variations and Change: New Challenges for Observing and Data Management Systems. *Journal of Climate* 6, 1481-1493.

5. Distributed Proofreaders. <http://www.pgdp.net/c/>

6. Bose, R. and Frew, J. 2005. Lineage Retrieval for Scientific Data Processing: A Survey. *ACM Computing Surveys* 37, 1-28.

7. Freire, J. and Davidson, S. 2008. Provenance and Scientific Workflows: Challenges and Opportunities. *ACM SIGMOD* 1345-1350.

8. Freire, J., Koop, D., Santos, E. and Silva, C. 2008. Provenance for Computational Tasks: A Survey. *Computing in Science and Engineering* 10, 11-21.

9. Simmhan, Y.L., Plale, B. and Gannon, D. 2005. A Survey of Data Provenance in e-Science. *ACM SIGMOD* 34, 31-36.

10. Simmhan, Y.L., Plale, B. and Gannon, D. 2005. A Survey of Data Provenance Techniques. *Technical Report TR-618.*

11. McPhillips, T., Bowers, S., Zinn, D. and Ludascher, B. 2009. Scientific workflow design for mere mortals. *Future Generation Computer Systems* 541-551.

12. Altinas, I., Barney, O. and Jaegar-Frank, E. Provenance Collection Support in the Kepler Scientific Workflow System. *Provenance and Annotation of Data. Lecture Notes in Computer Science* 4145. Springer, 2006.

13. Bowers, S., McPhillips, T., Riddle, S., Anand, M. and Ludascher, B. 2008. Kepler/pPOD: Scientific Workflow and Provenance Support for Assembling the Tree of Life. *IPAW '08.*

14. Missier, P., Behajjame, K., Zhao, J., Roos, M. and Goble, C. Data lineage model for Taverna workflows with lightweight annotation requirements.

15. Howe, B., Lawson, P., Anderson, E., Santos, E., Freire, J., Scheidegger, C., Baptista, A. and Silva, C. 2008. End-to-End eScience: Integrating Workflow, Query, Visualization, and Provenance at an Ocean Observatory. *IEEE International Conference on e-Science.*

16. Scheidegger, C., Koop, D., Santos, E., Vo, H., Callahan, S., Freire, J. and Silva, C. 2007. Tackling the Provenance Challenge One Layer at a Time. *Concurrency and Computation: Practice and Experience* 1.

17. Moreau, L. The Open Provenance Model (v1.01).

18. Moreau, L., Ludascher, B., Altinas, I., Barga, R.S., Bowers, S., Chin, G., Coen, S., Cohen-Boulakia, S., Clifford, B., Davidson, S. and Deelman, E. 2000. The First Provenance Challenge. *Concurrency and Computation: Practice and Experience.*

19. Osterweil, L.J. 1987. Software Processes Are Software Too. In *Proceedings of the 9th International Conference on Software Engineering.* Monterey, CA, 1987, 2-13.

20. LASER Lab. <http://laser.cs.umass.edu/>.

21. LASER Process Working Group. Little-LIL 1.5 Language Report. 3 October 2006. Laboratory for Advancement of Software Engineering Research, University of Massachusetts, Amherst.

22. Osterweil, L.J., Clarke, L., Ellison, A.M., Podorozhny, R., Wise, A., Boose, E. and Hadley, J.L. Experience in Using a Process Language to Define Scientific Workflow and Generate Dataset Provenance.

23. Osterweil, L.J. 1987. Software Processes Are Software Too. In *Proceedings of the 9th International Conference on Software Engineering.* Monterey, CA, 1987, 2-13.

24. Osterweil, L.J., Clarke, L., Ellison, A.M., Boose, E., Podorozhny, R. and Wise, A. 2009. Applying Software Engineering Technology to Support the Clear and Precise Specification of Scientific Processes.

25. Ellison, A.M., Osterweil, L.J., Clarke, L.,Haldley, J.L., Wise, A., Boose, E., Foster, D.R., Hanson, A., Jensen, D., Kuzeja, P., Riseman, E. and Schultz, H. 2006. Analytic webs support the synthesis of ecological data sets. *Ecology* 86, 1345-1358.

26. Boose, E., Ellison, A.M., Osterweil, L.J., Clarke, L., Podorozhny, R., Hadley, J.L., Wise, A. and Foster, D.R. 2007. Ensuring reliable datasets for environmental models and forecasts. *Ecological Informatics 2* 237-247.

27. Boose E. 2007. Prospect Hill Hydrological Stations. Harvard Forest Data Archive: HF070. <http://harvardforest.fas.harvard.edu:8080/exist/xquery/data.xq?id=hf070>

28. National Science Foundation (2010, May 10). Scientists seeking NSF funding will soon be required to submit data management plans. ScienceDaily. Retrieved May 11, 2010, from < http://www.sciencedaily.com/releases/2010/05/100510105136.htm>