Kepler/pPOD: Scientific Workflow and Provenance Support for Assembling the Tree of Life*

Shawn Bowers[1], Timothy McPhillips[1], Sean Riddle[1],
Manish Anand[2], Bertram Ludäscher[1,2]

[1]UC Davis Genome Center, University of California, Davis
[2]Department of Computer Science, University of California, Davis

Abstract. The complexity of scientific workflows for analyzing biological data creates a number of challenges for current workflow and provenance systems. This complexity is due in part to the nature of scientific data (e.g., heterogeneous, nested data collections) and the programming constructs required for automation (e.g., nested workflows, looping, pipeline parallelism). We present an extended version of the Kepler scientific workflow system to address these challenges, tailored for the systematics community. Our system combines novel approaches for representing scientific data, modeling and automating complex analyses, and recording and browsing associated provenance information.

1 Introduction

The National Science Foundation's Assembling the Tree of Life (AToL) initiative funds systematists investigating the phylogenetic (i.e., evolutionary) relationships of groups of organisms, with the ultimate goal of reconstructing the evolutionary origins of all life. AToL projects range from the study of the phylogeny of particular sets of organisms (e.g., using morphologial features or sequencing the genetic material of specimens) to the development of new computational approaches. Success of the AToL program, however, also depends on addressing a number of significant informatics challenges. For instance, there is no straightforward way to integrate data collected by different AToL projects, to test hypotheses against all data collected so far, or to begin to reconstruct the entire Tree of Life based on AToL data and results. Furthermore, the details of how the results of these projects were obtained—from the observations of specimens through the inference and evaluation of phylogenetic relationships—are difficult to determine in general. Not only is the provenance of specimens, observations, and computed results hidden within the data management infrastructure of each AToL project, many of the details required to reconstruct how results were obtained or to trace them back to primary observations are not recorded reliably.

The recently-funded pPOD project[1] aims at addressing these informatics challenges by developing (1) a common data model encompassing the data types used in the various AToL projects; and (2) methods for recording information about specimens, and relating this data and metadata to the results of phylogenetic analyses. For the latter, our

* This work supported in part through NSF grants IIS-0630033, OCI-0722079, and IIS-0612326.
[1] http://www.phylodata.org

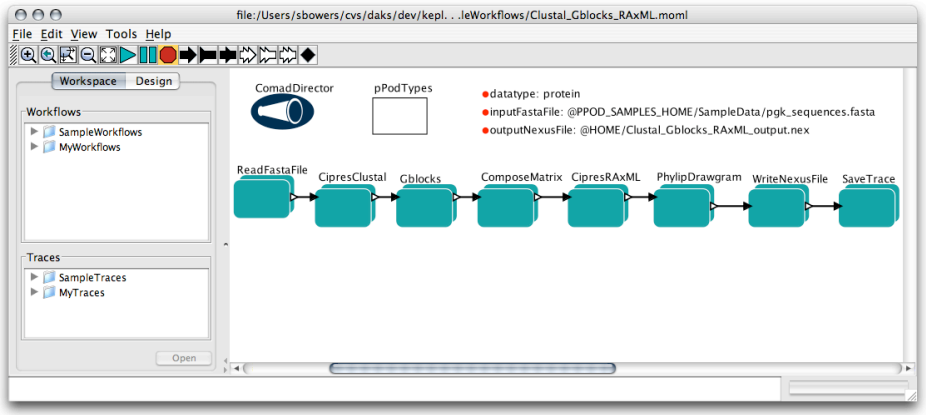

Fig. 1. A workflow in Kepler/pPOD with actors to read and parse FASTA files, compute multiple sequence alignments using Clustal, eliminate poorly aligned regions with Gblocks, create character matrices, infer phylogenetic trees using RAxML, draw resulting trees, and save outputs.

aim is to provide a mechanism to record and maintain a continuous processing history for all data and computed results across multiple analysis steps. These steps are often carried out using a wide variety of scripts, standalone applications, and remote services. This paper reports on our solution to this problem, i.e., of recording the provenance of results derived using heterogeneous software systems for phylogenetic data analysis.

Kepler/pPOD[2] is an extension of the Kepler scientific workflow system [7] for automating phylogenetic studies, orchestrating and routing data between invocations of local applications and remote services, and tracking the dependencies between input, intermediate, and final data objects associated with workflow runs. Kepler/pPOD uses the COMAD workflow design paradigm [9], which has built-in support for processing nested data collections in an assembly-line manner, complex dataflow constructs such as loops and subworkflows, and an efficient, fine-grained method for capturing and representing comprehensive data provenance. Thus, COMAD is well-suited for automating phylogenetic workflows, often yielding simpler and more reusable workflow designs [10] when compared with existing approaches [8,13,2,1] (e.g., that often employ "adapters" or "shims" between actors). In the remainder of this paper, we describe Kepler/pPOD, focusing on its use of COMAD and its support for recording, representing, and navigating provenance information.

2 The Kepler/pPOD System

Kepler/pPOD is a customized distribution of the Kepler scientific workflow system designed specifically to support phylogenetic data analysis. The goal of the current version of the system is to provide an easy-to-use desktop application that allows researchers to create, run, and share phylogenetic workflows as well as manage and explore the

[2] Kepler/pPOD can be downloaded at http://daks.ucdavis.edu/kepler-ppod

provenance of workflow results.[3] The main features of the system include: a library of reusable workflow components (*actors*) for aligning biological sequences and inferring phylogenetic trees; a graphical workflow editor (via Kepler) for viewing, configuring, editing, and executing scientific workflows (Fig. 1); a data model for representing phylogenetic data (sequences, character matrices, and trees) that facilitates conversion among different data and file formats; an integrated provenance recording system for tracking data dependencies created during workflow runs; and an interactive provenance browser for viewing and navigating data and actor-invocation dependencies.

Kepler/pPOD includes a number of sample workflows for phylogenetic analyses. These can easily be modified by changing parameters, selecting different input data, or substituting different methods for particular analysis steps. These workflows also demonstrate a variety of actors that provide access to both remote (web) services[4] and local applications. Fig. 1 shows one of the sample workflows in Kepler/pPOD.

2.1 The Computation Model of Kepler/pPOD

Kepler uses a graphical block-diagram metaphor for representing workflow specifications (i.e., workflow graphs). Blocks represent *actors* that carry out particular steps in an analysis, and connections among blocks represent dependencies between actor invocations. Kepler distinguishes between the workflow graph and the *model of computation* (MoC) used to interpret and enact the workflow. Workflow authors explicitly select a MoC by choosing a *director* (Fig. 1), which specifies whether a workflow is scheduled as, e.g., a process network (PN) or a synchronous dataflow network (SDF) [6]. Most Kepler actors used in PN or SDF workflows are *data transformers*, which consume input tokens and produce *new* output tokens on each invocation.

Kepler/pPOD includes a new director for *collection-oriented modeling and design* (COMAD) workflows, where actors and their connections are significantly different from those in PN or SDF. Instead of assuming that actors transform all input data to output data, COMAD employs an **assembly-line** processing style: COMAD actors (*coactors*, or *actors* for short) can be thought of as workers on a virtual assembly line, each contributing its part to the construction of the workflow products. In a physical assembly line, workers only "pick" relevant parts from the conveyer belt, letting irrelevant parts pass by for downstream processing. Coactors work analogously, recognizing and operating on data relevant to them as specified by a *read scope* parameter, adding new data products to the data stream, and allowing irrelevant data to pass through undisturbed. Thus, unlike actors in other workflow systems, actors are *data preserving* in COMAD where data *flows through* serially connected coactors rather than being consumed and produced at each stage.

An advantage of the assembly-line approach of COMAD is that one can put information into the data stream that could be represented only with great difficulty in traditional PN or SDF workflows. For example, COMAD embeds special tokens within the data stream to delimit collections of related data tokens. Because these *delimiter tokens* are paired, much like the opening and closing tags of XML elements (as

[3] Note that it is also possible to run the Kepler workflow engine separately from the workflow editor, allowing Kepler/pPOD to also support additional deployment configurations.

[4] e.g. CIPRes RESTful services, http://www.phylo.org/sub_sections/portal/

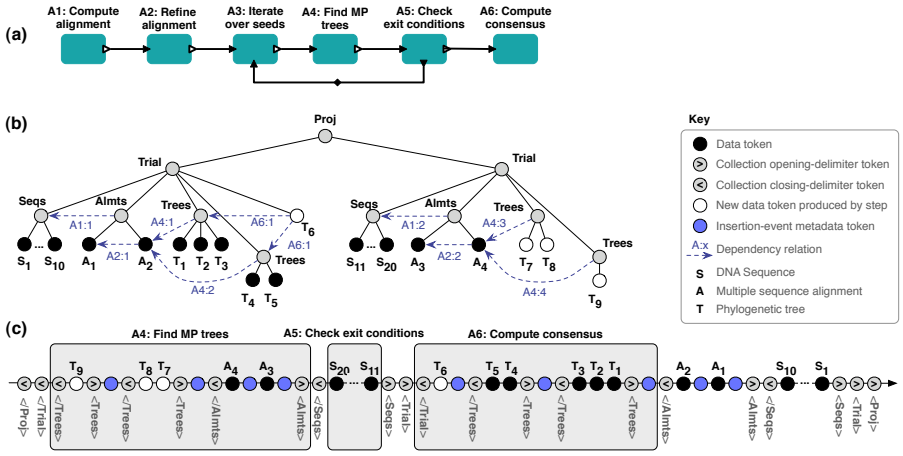

Fig. 2. A snapshot of a workflow run: (a) example workflow; (b) logical organization of data at a point in time during the run; (c) tokenized version of the collection structure where three actors are working *concurrently* on different parts of the data stream. Nested collections organize and relate data objects from domain-specific types (DNA sequences, alignments, phylogenetic trees). A *Proj* collection containing two *Trial* sub-collections is used to pipeline multiple sets of input sequences through the workflow. Provenance events (insert-data, insert-collection), insertion dependencies, and deletions (from the stream) are added directly as metadata tokens to the stream (c).

shown in Fig. 2), collections can be nested to arbitrary depths. This generic collection-management scheme allows actors to operate on collections of elements as easily as on single data tokens. Similarly, *annotation tokens* can be used to represent metadata for collections or individual data tokens, or for storing within the data stream the provenance of items inserted by coactors (see Fig. 2). The result is that coactors effectively operate not on isolated sets of input tokens, but on well-defined, information-rich collections of data organized in a manner similar to the tree-like structure of XML documents.

Another advantage of COMAD, compared to conventional dataflow approaches, is that COMAD workflows are generally more robust to change and easier to understand. For instance, inserting a new actor into a workflow or replacing an existing actor with a new version is straightforward, i.e., structural modifications to other parts of the workflow are not required. Similarly, while in traditional approaches maintaining collections of data and routing data to multiple actors requires the use of low-level control-flow constructs and associated actor connections, the same workflow in COMAD is often linear (as in Fig. 1). Thus, the intended function of a workflow can often be more easily understood, e.g., simply by reading the actor names in "assembly line order." Kepler/pPOD includes a library of coactors from which systematists can easily compose new workflows with minimal effort. For the same reason, the sample workflows in Kepler/pPOD can be easily modified to employ alternative methods for particular steps, extended with additional analysis steps, and concatenated with other workflows.

Fig. 2 illustrates a number of details of the COMAD approach, showing the state of a COMAD run at a particular point in time, and contrasting the logical organization of the data flowing through the workflow in Fig. 2(b) with its tokenized realization at the same point in time in Fig. 2(c). This figure also illustrates the pipeline-concurrency capabilities of COMAD by including two independent sets of sequences in a single run.

2.2 Recording and Representing Provenance in Kepler/pPOD

The provenance model used in Kepler/pPOD is unique (e.g., compared to [1,12,11,3,2]) in that it takes into account nested data collections, pipeline parallelism (in addition to the usual task parallelism), and actor scope expressions. The latter capture which parts of the data stream are visible for an actor and which parts of the output are created from them. In particular, no actor output can depend on items *outside* the actor's read scope. Our primary goal is to capture the information necessary to reconstruct and effectively present the derivation history of scientific data products, thereby supporting the main provenance needs of scientists. Thus, our approach is also different from those that focus on supporting workflow development and optimization by recording a detailed log of workflow events (e.g., execution time, invocation time-stamps, resources used); but similar in this way to efforts focusing on "scientist-oriented" provenance [12,3].

Recording Provenance Events. Provenance is captured during a workflow run by coactors, each of which places special provenance tokens directly into the token stream (Fig. 2) as needed to record its actions. Three types of provenance tokens are used to represent distinct provenance-related events during workflow execution. Actors add *insertion tokens* to the stream for each new data and collection item they produce. An insertion token consists of (1) the actor *invocation-id* used to produce the (data or collection) item; (2) the *token-id* of the produced item; and (3) a set of *token dependencies*, i.e., the collection- and data-token identifiers within the read-scope that contributed to the insertion of the item and that were input to the actor invocation. Similarly, actors add *deletion tokens* for each data and collection item they remove from the stream. A deletion token consists of (1) the actor invocation-id that removed the data or collection item; and (2) the token-id of the item that was removed. To maintain the dependencies between data products, our system simply tags removed items as deleted, preventing downstream actors from using deleted items, while retaining these items in the stream for later use by the provenance system.

Kepler/pPOD exploits nested data collections to help minimize the number and size of provenance tokens added to the token stream. In particular, insertion and deletion events are recorded at the highest node in the tree where they apply, and implicitly cascade to collection descendents. Insertion dependencies are applied in a similar way. For example, in Fig. 2, each alignment in an *Almnts* collection implicitly depends on each sequence in the corresponding *Seqs* collection, as indicated by the dependency between the two collections. In certain cases, actors also add *invocation-dependency* tokens to the stream that specify an ordering relation between two actor invocations. Specifically, invocation dependencies are added when actor invocations insert data or collection items that depend on a collection that was modified by a previously invoked actor. Here, a collection is considered modified when a data or collection node is added or removed (either a direct child or a descendent).

The result of running a Kepler/pPOD workflow is represented in an execution *trace*, an XML representation of the data and collections input to and created by a workflow run, the parameter values used for configuring actors, and each of the provenance tokens added by actor invocations. Each trace is assigned a unique id by the system, and trace files are organized in the workspace according to their corresponding workflows. Traces created from previous runs can also be used as input to workflows. In such cases, the system creates a new trace for the new run, referencing the input trace and thus linking to the provenance of the previous run. Kepler/pPOD can be used in this way to capture data dependencies across multiple workflow runs.

Constructing Provenance Graphs. Two types of provenance graphs are computed for displaying provenance information within Kepler/pPOD. These graphs are constructed directly from execution trace files. An *actor invocation graph* consists of actor-invocation nodes and directed invocation-dependency edges. An edge `B:1` $\rightarrow$ `A:1` in an invocation graph states that the first invocation of actor `A` was (logically) invoked prior to the first invocation of `B`, implying that `A:1` produced an item used by `B:1`, or more generally, `A:1` modified a collection used by `B:1`. A *data dependency graph* consists of nodes representing data items and directed edges representing insertion dependencies. Each edge is additionally labeled by the corresponding insertion invocation. An edge labeled `A:1` from D_3 to $\{D_1, D_2\}$ states that data item D_3 was produced by the first invocation of actor `A` from data items D_1 and D_2. Thus, D_1 and D_2 were "input" to `A:1`, i.e., they were within the read scope of the invocation. Data dependency graphs in Kepler/pPOD can distinguish items that depend only on a subset of the data input to an invocation. This is often the case, e.g., for actors that implement non-strict (i.e., "streamable") functions such as sliding-window algorithms.

The COMAD provenance model can be used to derive additional information, e.g., the set of collections input to a workflow run can be determined by selecting the collection and data items that were not inserted by actors and by removing any deletion tags. Similarly, the structure of a collection can be recreated at different points in the execution history, e.g., before or after a given set of actor invocations.

2.3 Displaying and Browsing Provenance in Kepler/pPOD

Within Kepler/pPOD, users can easily browse and navigate execution traces. The provenance browser, shown in Fig. 3, can be run directly from within Kepler/pPOD (e.g., by opening a trace file), or alternatively as a standalone application. The left-side of the browser displays the data, collections, and actor invocations of the workflow run, as well as a simple HTML navigation pane that displays details about these items. The browser also displays three different graphical views of the execution trace: (1) the *dependency history*, which combines the data-dependency and actor-invocation graphs; (2) the *collection history*, which shows how the various collections of a run were constructed (Fig. 4); and (3) the invocation graph (Fig. 4). Users can select and display the details of each item in a view (including the underlying data represented by a token, e.g., the particular sequence alignment or phylogenetic tree), and all of the views are synchronized. For instance, the selection of a data item in the dependency history also selects the corresponding item in the collection history. Using the browser, users can

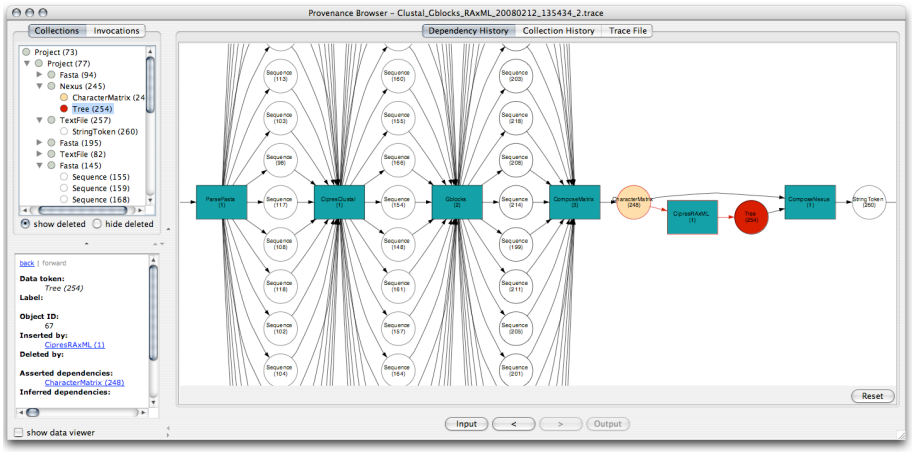

Fig. 3. The provenance browser of Kepler/pPOD showing the *integrated* dependency graph for a run of the workflow specified in Fig. 1.

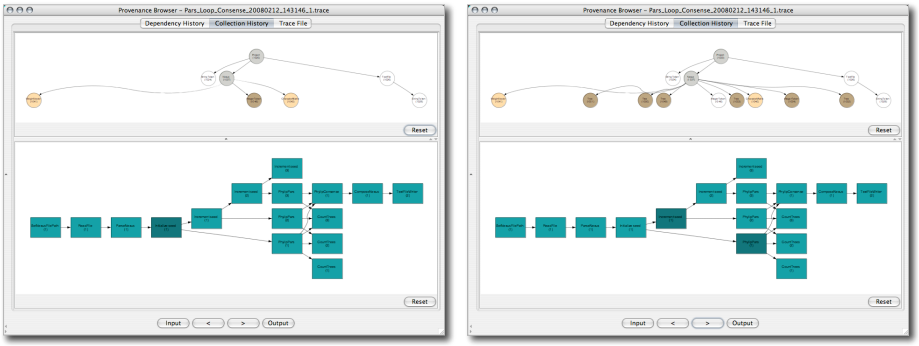

Fig. 4. The provenance browser showing collection and invocation history for a run of a workflow similar to Fig. 2(a): The resulting collections after the first invocation of an *Initialize Seed* actor (left); and the collection structure and invocation graph resulting from advancing one step through the execution history (right).

also incrementally step forward and backward through execution history, incrementally displaying (i.e., revealing or hiding elements, depending on navigation direction) the collection and data-dependency histories. This feature allows users to start from the input of the workflow and incrementally move forward through actor invocations to the final output. Similarly, it is possible to start at the output and navigate to the input, as well as move forward or backward at any point in between. The views in Fig. 4 are especially useful for analyzing how the structure of collections evolved throughout a workflow run; whereas the view of Fig. 3 more explicitly shows the steps and dependencies involved in generating data products.

3 Conclusion and Future Work

Kepler/pPOD supports the automation of phylogenetics workflows and the recording and visualization of data provenance for individual workflow runs. The system combines and implements our previous work on COMAD [9] and provenance [4], together with the new application presented here for browsing provenance traces and incrementally navigating execution histories. AToL projects will involve many interrelated workflows, where data produced during workflow runs commonly will be used as input to subsequent runs of different workflows, and workflows will be run multiple times with different parameterizations and on different input data sets. These projects also include tasks that cannot be fully automated between workflow runs, and the provenance of data products must be tracked across such manual data management tasks. We plan to extend Kepler/pPOD with *project histories* [5] for tracking data dependencies across multiple workflow runs and accommodating data management activities performed between runs. This will allow AToL researchers to organize their projects and data as they desire, while maintaining a continuous record of how results were obtained via a combination of manual operations and automated scientific workflows.

References

1. I. Altintas, O. Barney, and E. Jaeger-Frank. Provenance Collection Support in the Kepler Scientific Workflow System. In *Intl. Provenance and Annotation Workshop (IPAW)*, 2006.
2. L. Bavoil, S. P. Callahan, C. E. Scheidegger, H. T. Vo, P. Crossno, C. T. Silva, and J. Freire. VisTrails: Enabling Interactive Multiple-View Visualizations. In *IEEE Visualization*, 2005.
3. O. Biton, S. C. Boulakia, and S. B. Davidson. Zoom*UserViews: Querying Relevant Provenance in Workflow Systems. In *VLDB*, 2007.
4. S. Bowers, T. M. McPhillips, and B. Ludäscher. Provenance in Collection-Oriented Scientific Workflows. *Concurrency and Computation: Practice and Experience*, 2007.
5. S. Bowers, T. M. McPhillips, M. Wu, and B. Ludäscher. Project Histories: Managing Data Provenance Across Collection-Oriented Scientific Workflow Runs. In *DILS*, 2007.
6. E. A. Lee and A. L. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(12), 1998.
7. B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice & Experience*, 18(10):1039–1065, 2006.
8. S. Majithia, M. S. Shields, I. J. Taylor, and I. Wang. Triana: A Graphical Web Service Composition and Execution Toolkit. In *ICWS*, 2004.
9. T. McPhillips, S. Bowers, and B. Ludäscher. Collection-Oriented Scientific Workflows for Integrating and Analyzing Biological Data. *3rd International Workshop on Data Integration in the Life Sciences (DILS'06)*, 2006.
10. T. McPhillips, S. Bowers, D. Zinn, and B. Ludäscher. Scientific Workflow Design for Mere Mortals. submitted for publication, 2008.
11. L. Moreau, J. Freire, J. Futrelle, R. McGrath, J. Myers, and P. Paulson. The Open Provenance Model. Technical Report 14979, University of Southampton, 2007.
12. L. Moreau and B. Ludäscher, editors. *Computation and Concurrency: Practice and Experience, Special Issue on The First Provenance Challenge*, volume 20(5). Wiley, 2008.
13. T. Oinn, *et al*. Taverna: Lessons in Creating a Workflow Environment for the Life Sciences. *Concurrency and Computation: Practice & Experience*, 18(10):1067–1100, 2006.