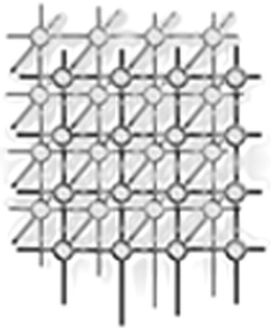
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2007; **00**:1–7 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

Provenance Trails in the Wings/Pegasus System[‡]



Jihie Kim[1], Ewa Deelman[1], Yolanda Gil[1],
Gaurang Mehta[1], Varun Ratnakar[1†]

[1] *Information Sciences Institute,*
University of Southern California,
4676 Admiralty Way,
Marina del Rey, CA 90292, U.S.A.

SUMMARY

Our research focuses on creating and executing large-scale scientific workflows that often involve thousands of computations over distributed, shared resources. We describe an approach to workflow creation and refinement that uses semantic representations to 1) describe complex scientific applications in a data-independent manner, 2) automatically generate workflows of computations for given data sets, and 3) map the workflows to available computing resources for efficient execution. Our approach is implemented in the Wings/Pegasus workflow system and has been demonstrated in a variety of scientific application domains. This paper illustrates the application-level provenance information generated Wings during workflow creation and the refinement provenance by the Pegasus mapping system for execution over grid computing environments. We show how this information is used in answering the queries of the First Provenance Challenge.

KEY WORDS: Semantic metadata, large scientific workflows, workflow validation, workflow mapping, workflow provenance, refinement provenance

Introduction

Our research focuses on supporting scientific applications that involve large amounts of computations operating on large volumes of data. As the number and size of computational jobs and data sources increase, the creation and the management of such workflows becomes impractical and even impossible without automatic generation, validation, and resource selection facilities. Our approach combines artificial intelligence and distributed computing techniques to create valid specifications of workflows of computations that can be efficiently executed in distributed shared resources. We use semantic representations to reason about application-level constraints and create valid execution-independent specifications of the

[†]E-mail: {jihie,deelman,gil,gmehta,varunr}@isi.edu
Contract/grant sponsor: National Science Foundation; contract/grant number: EAR-0122464,SCI-0455361

Received 29 November 2006
Revised 8 March 2007
Accepted

workflows. These workflow specifications are then mapped to the execution environment taking into account efficiency and dynamic availability of resources. Our approach is implemented in the Wings/Pegasus framework [7, 5, 2]. Wings uses semantic representations to reason about application-level constraints, generating not only a valid workflow but also detailed application-level metadata and provenance information for new workflow data products. Pegasus maps and restructures the workflow to make its execution efficient, creating provenance information that relates the final executed workflow to the original workflow specification.

This paper describes our approach to generating provenance information in the Wings/Pegasus framework. Our provenance capabilities produce 1) *application-level provenance* through the semantic representations used in Wings, and 2) *execution provenance* through the Pegasus workflow mapping process. We illustrate the provenance information generated by Wings during workflow instantiation and the refinement provenance by the Pegasus mapping system for execution over grid computing environments. We show how this information is used in answering the queries of the First Provenance Challenge [9] and suggest additional provenance queries supported by the Wings/Pegasus framework. We use the example workflow from the First Provenance Challenge (an fMRI workflow) in presenting our approach. All the representations of the workflow and provenance data are available in the Wings/Pegasus provenance site [14].

Wings/Pegasus: Creating and Executing Large Workflows

To support the creation and validation of very large workflows, we have developed an approach that considers three stages for workflow creation, where each stage corresponds to a different level of abstraction, and where new type of information is being added to the workflow. Figure 1 illustrates the process of workflow creation. The same diagrams in a larger size are available in [5]. Wings supports the first two layers, while Pegasus supports the third.

The first layer of workflow creation defines *workflow templates* that are data- and execution-independent specifications of computations. Workflow templates express repetitive computational structures in a compact manner and identify the types of components to be invoked and the data flow among them. A workflow template is an abstract specification of a workflow, with a set of nodes and links where each node is a placeholder for a component or component collections (for iterative execution of a program over a file collection), and each link represents how the input and output parameters are connected. It includes the type of data that the workflow is designed to process, but the specific data to be used are not described in the template. A workflow template can be shared and reused among users performing the same type of analysis.

The second layer of workflow creation uses workflow templates as a starting point to create *workflow instances* that are execution-independent. Workflow instances specify the input data needed for an analysis in addition to the application components to be used and the data flow among them. A workflow instance can be created by selecting a workflow template that describes the desired type of analysis and binding its data descriptions to specific data to be used.

The third and final layer of workflow creation maps workflow instances onto executable workflows. Executable workflows are created by taking workflow instances and assigning actual

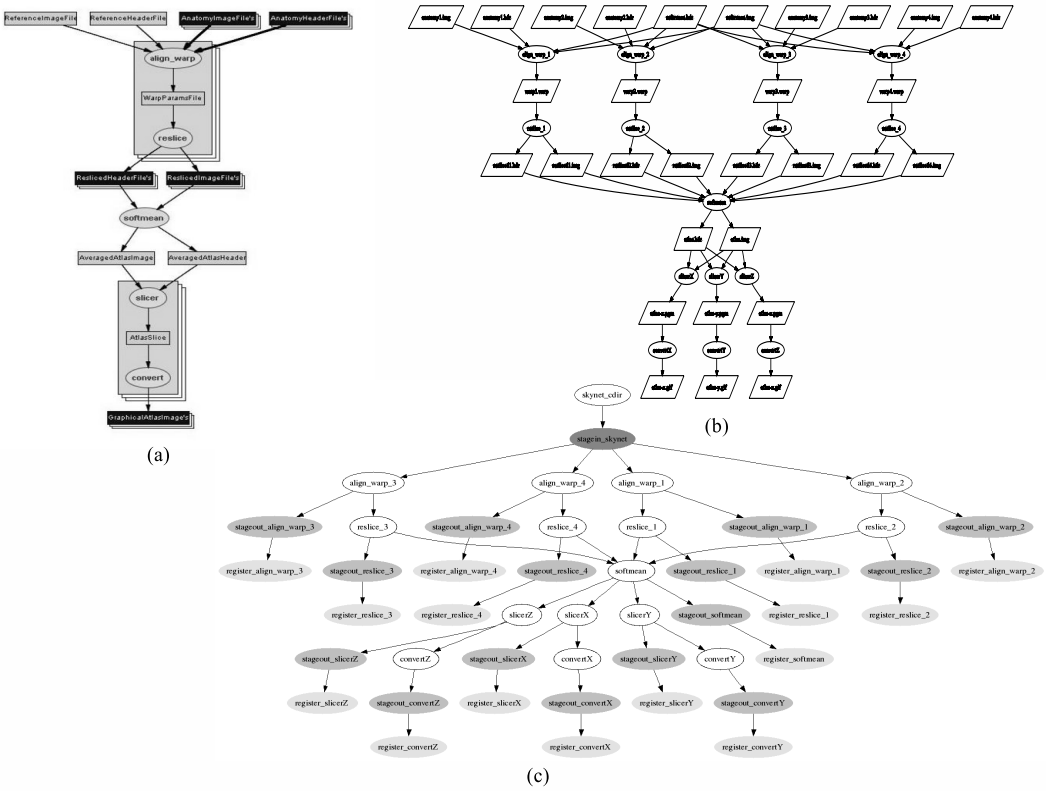


Figure 1. Layers for workflow creation: (a) Layer 1 - workflow template (generic recipe), (b) Layer 2 - workflow instance (data specific), and (c) Layer 3 - executable workflow (actual run, files are not shown). Ovals are tasks and rectangles show data in the workflows.

resources that exist in the execution environment and reassigning them dynamically as the execution unfolds. Executable workflows fully specify the resources available in the execution environment (e.g., physical replicas, sites and hosts) that should be used for execution. This stage is performed by Pegasus.

Wings implements the first two layers of workflow creation, from workflow templates to workflow instances. Pegasus transforms a workflow instance into an executable workflow through a mapping that assigns workflow tasks to available grid resources for execution.

Generating Application-Level Provenance during Workflow Creation

This section describes our approach for representing and reasoning about workflow constraints. We show how Wings represents constraints on file collections and their elements, constraints on inputs and outputs of each component, and global constraints among multiple components.

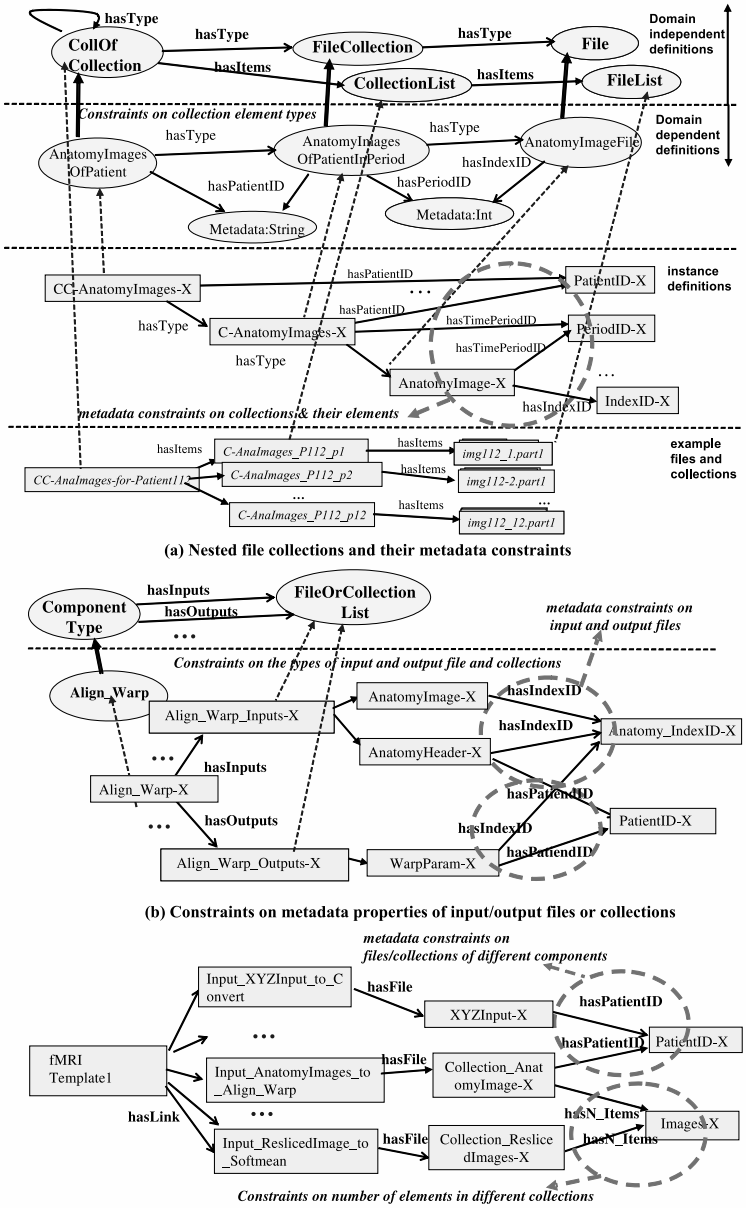


Figure 2. Semantic metadata constraints in Wings/Pegasus.

These constraints are used during workflow creation to generate application-level provenance. Wings uses the Web Ontology Language (OWL) from W3C [10] for representing this provenance information.

The first type of metadata constraints is on individual files and nested collections. Each file class can have one or more metadata properties associated with it. Metadata of a file describe what the file contains, how it was generated, etc. For example, an AnatomyImage file has hasIndexID that represents what it contains. These are represented in OWL and stored in the *file library* (FL). The creation of a *Brain Atlas* needs several anatomy image files, and several Brain Atlases can be created for each patient over different time periods. That is, the anatomy image files for a patient are naturally structured as a collection of files. In our ontology, the concept collection represents both simple file collections and nested collections. Each collection should specify the type for the collection element using the *hasType* property. There can be constraints between a collection and its elements. For example, for an image collection for a time period, the PeriodID for an individual image file should be the same as the PeriodID of the collection, as shown in Figure 2 part(a).

The second type of metadata constraints is on components and their inputs and outputs. In the *component library* (CL), each workflow component is described in terms of its input and output data types. In Figure 2 part(b), the Align_Warp component has two inputs: an AnatomyImage file and an AnatomyHeader file. Each AnatomyHeader file has an IndexID that should be the same as the IndexID of the AnatomyImage file. Given these inputs, the Align_Warp component produces a WarpParam file. The metadata for the generated WarpParam file depends on the metadata of the inputs. In the above example, the IndexID and the PatientID of the AnatomyImage are propagated to corresponding metadata properties of the output WarpParam file when Wings generates the workflow instance.

The third type of constraints is on different components and files in templates. These constraints are global to the workflow. First of all, the components should use image data for the same patient. In Figure 2 part(c), the PatientID of the XYZInput file used in the Convert step should be the same as the PatientID of the collection Collection_AnatomyImage of AlignWarp. (We use a isSameAs property in representing equalities of metadata.) In addition, the components should use the same number of images throughout the workflow.

The Wings algorithm for reasoning about these constraints while creating workflow instances is described in detail in [7]. The system starts with a template in the *workflow library* (WL) and the metadata of the initial input data. Wings identifies dependencies among the data based on metadata constraints, and propagates and creates metadata using the component constraints and the workflow template constraints. The created workflow instances are stored in WL.

Generating Execution Provenance during Workflow Refinement and Execution

The workflow instance from Wings is sent to Pegasus for mapping onto the available execution resources. This mapping process consists of various refinement operations. First, the workflow instance can be partitioned into several smaller *sub-workflows*. Pegasus can run further refinement operations on each individual sub workflow. Figure 3 shows an example of the Pegasus refinement process. The original workflow ($workflow1$) is partitioned into

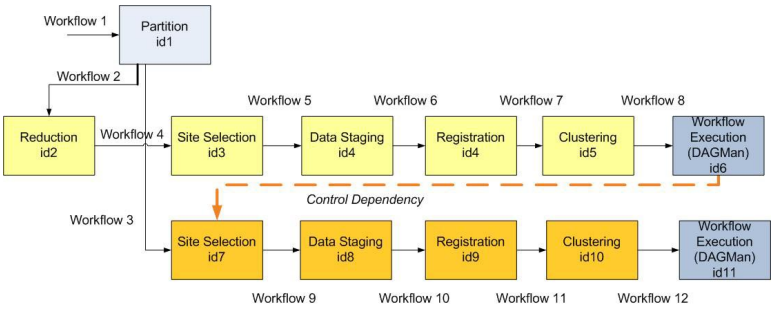


Figure 3. Workflow refinement: executable partitions are mapped onto resources.

two subworkflows ($workflow2$ and $workflow3$). Since in this example, there is a dependency between the two sub-workflows, $workflow2$ is refined first. A reduction is performed where unnecessary tasks in the given workflow are pruned out. This is applicable when datasets to be generated by these tasks have already been previously computed and it is more efficient to access them than to recompute them. Next, the *site selection* is performed where tasks in the workflow are mapped to available resources. The next two refinement steps augment the workflow with data management steps, first by adding data stage-in and stage-out jobs that transfer the data to and from where the computations are to take place, and then by adding data registration jobs that register data in various catalogs so that it can be subsequently found. Jobs scheduled for execution on the same resource can be clustered together to improve the overall workflow performance. Thus, the final refinement step performs this optimization. The original sub-workflow $workflow2$ results in $workflow8$ which can be submitted to the Condor DAGMan execution engine[3]. Upon the successful execution of this part of the original workflow, the following subworkflow $workflow3$ is refined in a similar series of steps.

If we look at the series of refinements conducted on the workflow, this is simply a series of transformations where the inputs and outputs are the workflows undergoing the refinement. For example, we apply the *site selection* transformation on $workflow4$ and obtain a $worfklow5$. The workflows, just as the data in the application workflow may have provenance associated with them, including information about which transformation generated them, on which system, with what performance and input data. We are currently working on a model that would uniformly treat the refinement provenance records and the application execution provenance records. One simple representation for these records could be the following structure: <object id> [[data I/O][function performed][performance info][optional annotations]]. The data I/O would define the inputs and outputs to a transformation. The function performed refers to the transformation. The optional annotations include a justification of the reasons for the tasks performed. Currently these refinement processes records are not captured in our provenance system but we are planning to implement an instrumentation facility in the near future.

Currently, the details of the job executions are recorded in the Virtual Data System(VDS)[13] *provenance tracking catalog* (PTC). Each job in the planned workflow is followed by a postscript

that stores information in the PTC. Information such as the job executable name and arguments, start time for execution and duration of each job, machine information on which the job ran etc are stored in PTC. For a failed job, the error message and the exit status are stored in PTC.

Answering Provenance Challenge Queries

The queries of the First Provenance Challenge are expressed as queries to the workflow library, component library, file library, and execution provenance catalog. The Wings/Pegasus capabilities can address all such queries. Table 1 shows how each query is mapped to queries to the Wings/Pegasus libraries and catalogs. Answers to a few of the queries (marked with asterisk) were not delivered, but the table shows how the query would be answered by information that is already captured in the current system. All the answers are available at the ISI provenance site [14]. Since Wings uses OWL for representing metadata, components, and workflows, the W3C Candidate Recommendation SPARQL [12] for RDF/OWL queries is used to express application-level provenance queries. Pegasus stores information on resource mappings and execution provenance in a database format and SQL is used to express execution provenance queries. An important observation is that provenance questions do not have to be pre-defined and can be answered on the fly by deriving the answer from the information that is recorded by Wings/Pegasus. This is the case with queries such as Q1, Q3, Q5, Q6, and Q8, which involved figuring out which component created the given file, which input files were used for the component, which component generated the input files, etc. Instead of recording this provenance information directly as metadata, Wings derives this information by navigating the structure of the associated workflow. The next section illustrates this by showing additional types provenance queries that Wings/Pegasus could answer based on the information currently recorded. Another important observation is that most of the queries of the First Provenance Challenge can be answered using information that is available in Wings/Pegasus before the execution of the workflow begins. For example Q2 only needs the template or an abstract description of the steps involved: "Find the process that led to Atlas X Graphic, excluding everything prior to the averaging of images with softmean." Such queries can be mapped to a sequence of SPARQL queries that extract relevant information from the template or one of the workflow instances. The following shows a SPARQL query that retrieves any collections that contain an Atlas X Graphic file:

```
SELECT DISTINCT ?url ?coll
FROM <http://www.isi.edu/ikcap/Wingse/domains/fMRI/fileLibrary.owl>
WHERE { flib:run1_atlas_x.gif flns:hasCreationMetadata ?url .
   OPTIONAL { ?coll flns:hasItems ?list .
              ?list apf:list flib:run1_atlas_x.gif .
              FILTER jfn:print( jfn:listIndex(?list, flib:run1_atlas_x.gif) ) }}
```

Most queries in the First Provenance Challenge concerned application-level provenance. Queries regarding provenance of input or output datasets can be answered by navigating the relevant workflow instance. For example the following query constrains the retrieval workflow instances based on a metadata property of output files (global maximum value): "Find all Atlas Graphic images outputted from workflows where at least one of the input Anatomy Headers had an entry global maximum=4095."

Table I. Answering Provenance Queries.(Provenance Information Used: FL - file library, CL - component library, WL - workflow library, PTC - Provenance Tracking Catalog)

Query	*Provenance*	*provenance queries involved*
Q1	FL WL	Find workflow instances that contain relevant files and traverse the workflow instances for extracting relevant portion.
Q2	FL WL	Find workflow instances that contain relevant files and traverse the workflow instances for extracting relevant portion.
Q3*	FL WL	Find workflow instances that contain relevant files and traverse the workflow instances for computing stage levels and extracting relevant portion.
Q4	PTC	Join of the invocation and job tables.
Q5	FL WL	Find relevant files in the file library and traverse the associated workflow instances for extracting relevant portion.
Q6	FL CL WL	Find workflow instances that contain relevant files and find relevant component instances, and traverse the workflow instances for extracting relevant portion.
Q7*	WL	Run graph comparison algorithm over the templates (expensive) and compare the bindings in the workflow instances.
Q8	FL CL WL	Find workflow instances that contain relevant files and find relevant component instances, and traverse the workflow instances for extracting relevant portion.
Q9*	FL CL WL	Retrieve the annotations and find workflow instances that contain relevant files and find relevant component instances, and traverse the workflow instances for extracting relevant portion.

```
SELECT DISTINCT ?url
FROM <http://www.isi.edu/ikcap/Wingse/domains/fMRI/fileLibrary.owl>
WHERE { ?arg flns:hasValue "4095" .
   ?file domflns:hasGlobalMaximum ?arg .
   ?file flns:hasCreationMetadata ?url }
```

Execution provenance queries are answered accessing the execution provenance catalog. An example of such a query is: "Find all invocations of procedure align_warp using a twelfth order nonlinear 1365 parameter model (see model menu describing possible values of parameter "-m 12" of align_warp) that ran on a Monday." The following SQL query provides that information:

```
SELECT i.creator, i.creationtime, i.start, i.duration, i.tr_name, i.resource, i.host, j.args,
j.exitcode FROM ptc_invocation i, ptc_job j WHERE i.tr_name="align_warp" and
i.wf_label="fmri" and i.id=j.id and j.args like "%m 12%" and DAYNAME(i.start)='Monday';
```

Additional Provenance Queries Answered by Wings/Pegasus

The First Provenance Challenge did not exercise some of the unique capabilities of Wings/Pegasus: 1) extensive semantic metadata created and propagated during the workflow creation process to describe workflow tasks and data products *before execution*, 2) detailed records of optimization transformations to prepare the workflow for execution that can explain why the workflow actually executed was different from the workflow originally submitted for execution. These are some provenance query types enabled by Wings/Pegasus:

1. *Queries on mutually constrained datasets in a workflow*: The system can query relations among the files used or produced by different components in a workflow. The following provides an example: "What are the collections that should have the same number of elements as the anatomy header file collection?"

```
SELECT ?coll_same_N_elements
FROM <http://www.isi.edu/ikcap/Wingse/domains/fMRI/templates/Template.owl>
WHERE {  fmrit:AnatomyHeaderCollection_1 flns:hasN_items ?n_imgs .
  ?coll_same_N_elements flns:hasN_items ?n_imgs}
```

2. *Queries to justify provenance information in datasets*: The system can query the value of a metadata property that is dependent on other existing properties. The following provides an example: "When the index value of an anatomy image is 2, what is the index value of the resulting warp parameter file?"

```
SELECT ?warp_param_file_idx
FROM <http://www.isi.edu/ikcap/Wingse/domains/fMRI/templates/run11_ExpandedInstance.owl>
FROM <http://www.isi.edu/ikcap/Wingse/domains/fMRI/fileLibrary.owl>
WHERE {
 ?link rdf:type wflns:InputLink . ?link wflns:hasFile ?afilec .
 ?link wflns:hasDestinationNode ?node . ?afilec rdf:type domflns:AnatomyImageCollection .
 ?afilec flns:hasItems ?list . ?list apf:list ?file_item .
 ?file_item domflns:hasIndexVal ?v . ?v flns:hasValue "2" .
 ?olink wflns:hasOriginNode ?node . ?olink wflns:hasFile ?wfilec .
 ?wfilec rdf:type domflns:WarpParamsCollection . ?wfilec flns:hasItems ?wlist .
 ?wlist apf:list ?wfile_item .
 FILTER (jfn:listIndex(?list, ?file_item) = jfn:listIndex(?wlist, ?wfile_item)) .
 ?wfile_item domflns:hasIndexVal ?wv . ?wv flns:hasValue ?warp_param_file_idx}
```

3. *Queries regarding execution and optimization transformations*: Job execution details are stored in the executable provenance catalog. The following provides simple example queries: "What is the total time taken for generating a given dataset?"

```
SELECT wf_label as workflow , sum(duration) as "duration(s)" FROM ptc_invocation
WHERE wf_label="fmri" group by wf_label;
```

Another example query regarding execution provenance is: "What is the total time taken to run each type of executable?".

```
SELECT tr_name as executable, count(id) as count, sum(duration) as duration(s)
FROM ptc_invocation WHERE wf_label="fmri" group by tr_name;
```

The overall execution times may be much shorter than expected, and provenance queries may be posed to find out the optimizations that led to such performance improvements. For example, follow up queries could be answered regarding why a given component was not executed, by walking through the transformations applied to the original workflow submitted for execution and pointing out the reductions applied.

Many more provenance queries can be composed on the fly and answered from the libraries and catalogs of provenance trails kept by the Wings/Pegasus workflow system. The integration of application-level and execution-level provenance information into a single query is possible since the system uses unique identifiers for the workflow templates, instances, and executables involved obtaining a given result.

Related Work

Various data provenance and metadata approaches have been developed, including the approaches that contributed to the Provenance Challenge [9, 11]. The metadata reasoning capabilities of most existing systems focus on files and simple collections but as the size and the complexity of the workflows increase, representation and management of nested collections are becoming more important[7, 1]. Existing work on analyzing dependencies among metadata is limited to validation of input data of individual components [15] or causal relations between events and data [8]. However, often there are global constraints on inputs and outputs

of multiple components. Semantic web techniques including OWL and RDF are used in other systems for representing semantic provenance [6, 16]. They use OWL/RDF reasoners to infer connections among files and processes in workflows in answering the Provenance Challenge queries. However, most of the existing metadata approaches focus on analyses of provenance information created during execution. The Pegasus/Wings framework has two distinct features over these approaches: detailed semantic metadata available before execution, and rich optimization trails that led to the workflow actually executed.

Summary and Conclusion

We have described our approach to the First Provenance Challenge queries, and proposed additional provenance queries that are supported by the Wings/Pegasus workflow system. In particular, we presented several novel provenance capabilities that result from the creation and propagation of semantic information about workflow components and data products before execution occurs, and the recording of optimization transformations performed on submitted workflows to improve execution performance.

REFERENCES

1. Bowers, S., McPhillips, T., Ludaescher, B. A Provenance Model for Collection-Oriented Scientific Workflows. In *Concurrency and Computation: Practice and Experience*, 2007.
2. Deelman, E., Singh, G., Su, M., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, B., Good, J. Laiety, A., Jacob J., Katz, D., Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems. In *Distributed Systems Scientific Programming Journal*, Vol. 13(3), 2005.
3. Frey, J., Tannenbaum, T., Livny, M., Foster, I., Tuecke., S., Condor-G: A Computation Management Agent for Multi-Institutional Grids., In *Cluster Computing*, Vol. 5, pp. 237-246, 2002.
4. Jena – Semantic Web Framework for Java. http://jena.sourceforge.net, 2007.
5. Gil, Y., Ratnakar, V., Deelman, E., Mehta, G., Kim, J., Wings for Pegasus: A Semantic Approach to Creating Very Large Scientific Workflows. In *The Eighteenth Conference on Innovative Applications of Artificial Intelligence*, Vancouver, BC, July 2007.
6. Golbeck, J., Hendler, J., A Semantic Web Approach to Tracking Provenance in Scientific Workflows. In *Concurrency and Computation: Practice and Experience*, 2007.
7. Kim, J., Gil, Y., Ratnakar, V. Semantic Metadata Generation for Large Scientific Workflows. In *Proceedings of the International Semantic Web Conference*, Atlanta, GA, November 2006.
8. Miles, S., Groth, P., Munroe, S., Jiang, S., Assandri T., Moreau, L., Extracting Causal Graphs from an Open Provenance Data Model, In *Concurrency and Computation: Practice and Experience*, 2007.
9. Moreau, L. et al., The First Provenance Challenge. In *Concurrency and Computation: Practice and Experience*, 2007.
10. OWL. http://www.w3.org/TR/owl-guide, 2007.
11. Simmhan Y., Plale B., Gannon, D. A Survey of Data Provenance in e-Science In *SIGMOD Record*, vol. 34, 2005.
12. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query, 2007.
13. VDS. http://vds.isi.edu, 2007.
14. Wings/Pegasus for the First Provenance Challenge. http://vtcpc.isi.edu/provenance, 2007.
15. Wong, S., Miles, S., Fang, W., Groth, P., Moreau, L. Validation of E-Science Experiments using a Provenance-based Approach. In *Proceedings of the 4th International Semantic Web Conference*, Galway, Ireland, November 2005.
16. Zhao, J., Goble, C., Stevens, R., Turi D., Mining Taverna's Semantic Web of Provenance. In *Concurrency and Computation: Practice and Experience*, 2007.