Wings for Pegasus: A Semantic Approach to Creating Very Large Scientific Workflows

Yolanda Gil[1], Varun Ratnakar [1], Ewa Deelman[1],
Marc Spraragen[1], and Jihie Kim[1]

[1] Information Sciences Institute, University of Southern California
4676 Admiralty Way, Marina del Rey CA 90292, United States
{gil, varunr, deelman, marcs, jihie}@isi.edu

Abstract. Scientific workflows are being developed for many domains as a paradigm to manage complex scientific computations. In our work, we are challenged with efficiently generating and validating workflows that contain large amounts (hundreds to thousands) of individual computations to be executed over distributed environments. We describe a new approach to workflow creation and validation that uses semantic representations to describe complex scientific applications in a data-independent manner, then automatically generates workflows of computations for given data sets, and finally maps them to available computing resources. We have implemented this approach in Wings and used it to create workflows of thousands of computations, which are submitted to the Pegasus mapping system for execution over grid computing environments.

Keywords: Workflow generation, semantic grid, scientific workflows, grid workflows, workflow editors.

1 Introduction

Scientific workflows are emerging as an effective paradigm to represent and manage complex scientific applications [10,19]. Scientific communities are sharing resources including data repositories, services, instruments, and computing resources [2,3,9,22,24]. Workflows provide an effective representation that captures how these very heterogeneous resources can be configured and assembled for a wide variety of purposes, and that facilitates the management of their execution in such distributed environments. As sharing of data and resources increases in scientific communities, the creation and management of workflows is central to the future of scientific analysis and computations.

Some scientific applications, notably in bioinformatics, are cast as workflows of web services in distributed environments. These applications require support to discover, describe, compose, and execute workflows that are mapped to web services available in the execution environment. Several workflow environments are successfully support scientists today in creating and managing these service-based workflows [18,21].

Many other scientific applications do not use workflows composed of services. Instead, workflows are composed of jobs that perform computations on remote hosts in distributed environments through remote job submissions, utilizing data that reside in catalogs that are replicated in the execution environment, and creating data products that need to be stored back in those repositories. The computations are nodes in the workflow, and the links in the workflow represent the data flow among computations. Data is typically available in files, and each file is described with *metadata* attributes [23] that describe its properties (e.g., its creation date, the instrument used for collection, or the computation that created it, and other domain-specific attributes). Examples of workflow systems of this kind are Pegasus [6,7,8,4,5,13] and Askalon [27], and more recently Kepler [18] as well. We refer to these workflows as *computational workflows*.

Creating and validating these computational workflows is a very challenging enterprise. Some of our prior work has addressed the design of intelligent workflow editors that assist users in creating valid workflows using AI planning techniques and semantic representations of workflow constituents [16]. However, in many scientific applications there is a need to scale up to data sets of thousands of elements.

The goal of our work is the creation and execution of large scientific workflows that include in the order of hundreds or thousands of computations. This paper presents a novel approach that exploits semantic representations of workflows to express repetitive computational structures in a compact manner, describe underspecified data collections, and process these representations to create workflows that can be then mapped to available resources for execution. We have implemented this approach in Wings, and integrated it with Pegasus into an end-to-end workflow creation and execution system.

We begin with a detailed motivation showing the nature of the complexity of creating large workflows of computations. After reviewing related work, we present our approach and its implementation in Wings. We finalize with a discussion and plans for future work.

2 Motivation: Creating Very Large Scientific Workflows

In our work, we distinguish three distinct stages in the creation of workflows [11]. The first stage is to create *workflow templates*, which specify the high-level structure of the workflow in a data-independent representation. The second stage is to create *workflow instances* (or workflows for short), which specify what data is to be used in the computation. Workflow instances are independent of execution resources, that is, they can be mapped to any execution environment by binding tasks to available resources. The third stage is to create an *executable workflow,* which specifies the data replicas to be used and their locations, the hosts where computation will occur, and the appropriate data movements across distributed locations. These three stages allow us to manage the complexity of workflow creation by making the process more modular.

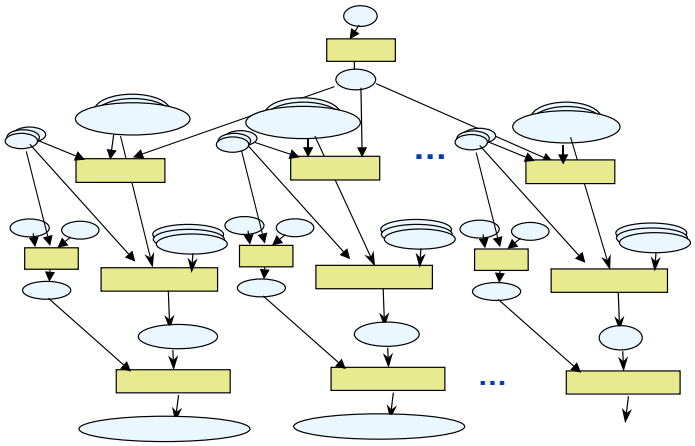
The later stage of creation of executable workflows is done by Pegasus [6,7,8,4,5,13]. Pegasus performs automated mapping of workflows to execution

resources and the management of their execution in distributed grid environments. Pegasus is now a production-level workflow mapping and execution engine that is being used in a variety of scientific applications and executes workflows in small, medium, and large size grid environments such as the Teragrid and the Open Science Grid [2,9]. What Pegasus users are currently lacking are general-purpose mechanisms to create and validate their very large workflows.

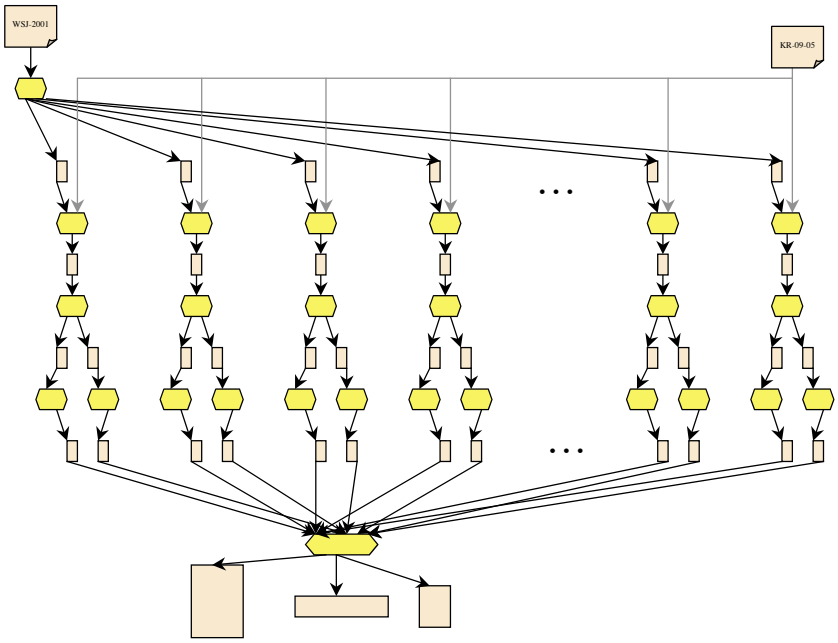
Many scientific applications require the processing of dataset elements one at a time with identical computations. Figure 1 shows on the left an example from our collaboration with the Southern California Earthquake Center (SCEC) [20] and on the right an example of a workflow from statistical natural language processing [17]. The first workflow conducts the same kind of simulation for all ruptures of all faults being considered. It can be iterated again for several sites. Computation results are usually re-compiled in later steps to provide a summary view back to the scientist. The second workflow reflects the parallel processing of a large corpus by breaking it up into smaller chunks. The results from each chunk are re-compiled in later steps. Both examples illustrate the regular structure that appears in many scientific workflows for processing large data sets. Similar kinds of structures have been shown, for example, in workflows for creating image mosaics of astronomical data [7] and for finding clusters of galaxies in the Sloan Digital Sky Survey [1]. In the coming years, workflows will continue to grow in data set size as well as complex interleaving of creation and execution. Providing assistance in creating and managing these large workflows will be not only desirable but absolutely necessary.

An on-going problem is that these kinds of workflows are created using ad-hoc scripts that specify each individual job, create meaningful identifiers for all new data products of the workflow, and weave the dataflow connections among the individual jobs. The iterations across data sets are managed implicitly in the scripts. If the workflow needs to be changed, or a new workflow needs to be created with some of the same components, new scripts have to be written. This process is not very practical and is highly prone to error, so it does not scale well as the workflow size increases. The validation of the resulting workflow is a challenge, mostly done by hand. There should also be a way to specify that two separate collections of data used as input to the workflow should have the same cardinality, whatever that cardinality is, in order for the workflow to be valid.

There are important issues regarding the coupling of the workflow creation and the workflow execution. Ideally, the workflow should be completely specified in terms of the kinds of computations to be performed and the kinds of data to be created, but be independent of the choice of hosts and other resources allocated for execution. It is desirable that the same workflow can be mapped at execution time to the resources that are available at that time, so this information should not be included in the workflow. The Pegasus workflow mapping engine performs that kind of mapping, binding data descriptions to one of many possible replicas, selecting hosts to execute the computations, moving data to where computation will occur, and moving data products to data repositories. This mapping engine can do a better job at reservation and provisioning of resources if the entire workflow structure is known ahead of execution. It is useful for this reason to have the ability to generate as complete a description of the workflow and its anticipated computations and data ahead of the execution process.



(a)



(b)

Fig. 1. Some examples of scientific workflows with hundreds or thousands of computations and data products, illustrating their regular structure in processing subsets of the data one at a time.

Execution requirements also influence the nature of the workflows. Because many failures can arise when executing each workflow component (insufficient memory, full file system, code bugs, etc), the system must manage the recovery from those failures and figure out what remains to be executed in the workflow. Workflows that have control constructs such as conditional branching and iteration are harder to manage because they require the execution system to have some persistent representation of their distributed execution state. For these reasons, the workflows that we consider are structured as directed acyclic graphs (DAGs) without control constructs.

In summary, our goal is to support the creation and execution of scientific workflows that include hundreds or thousands of computational steps. This requires:

1. creating workflow descriptions that orchestrate large amounts of computations while ensuring that that are valid in that they respect data and computation constraints,
2. handling data sets with many elements and manage the creation of iterative substructures in the workflow that process each of those elements,
3. generating appropriate metadata descriptions for all the new data sets created during execution, and specifically full elaboration of workflow descriptions are required in order to submit for execution so the system can detect pre-existing intermediate data and avoid unnecessary recomputation.

3 Approach

To support the creation and validation of very large workflows, we have developed a new approach to represent and reason about workflows and their data so that:

- **workflow templates and instances are semantic objects**, where all their components, links, data requirements, and data products are represented in ontologies with appropriate constraints among them,
- **data collections are specified with intensional descriptions in workflow templates and extensional descriptions in workflow instances**, with appropriate relationships as a data set is concretely specified during the creation of a workflow instance from a workflow template,
- **intensional descriptions of node collections that offer appropriate abstractions for the repetitive structure of the workflows** at the template level that can support reasoning about expanding those structures at the instance level once the data sets are specified

There are several important benefits of this approach. By making the description of a workflow template very compact through intensional descriptions of data sets and node sets, it is easier to create the basic structure of the workflow and validate it with smaller data sets. By specifying declaratively data collections and their constraints

and properties we can validate the input data as well as intermediate data products of the workflow.

The next section describes our current implementation of this approach and illustrates the main ideas through examples.

4 Wings for Pegasus

Wings implements the approach outlined above by supporting the creation of workflow templates and instances, which are then submitted to Pegasus to create executable workflows. Figure 2 gives an overview of the system. Notice that workflow templates and instances can be created by different users. Experienced scientists can create templates that comply with widely-accepted analyses that reflect valid scientific methodologies. Less experienced users can perform many analyses with different kinds of data by creating and executing workflow instances. A Composition Analysis Tool (CAT) assists users during template creation by checking that the template is valid and making suggestions based on the constraints and definitions in the ontologies. An early version of this tool can be found in [16].

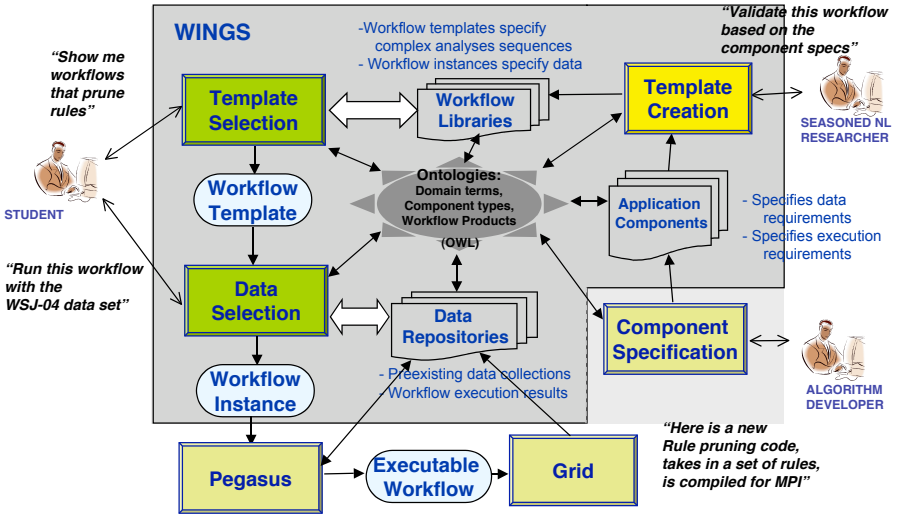


Fig. 2. Overview of the Wings/Pegasus Architecture.

In Wings, workflow templates and instances are semantic objects and so are their components (nodes), the links among them, and the data generated by workflows. All are accessible in a central repository, although we envision more distributed approaches in future versions (e.g., [23]). We use OWL-DL as the representation language, and Jena as the underlying reasoner.

Data is represented as individual files that can be grouped into file collections. Nested collections are also supported. All the items within a collection must have a common type. The core definitions of the file ontology are:

- **File**: Represents the basic File class.
- **DataCollection** is used to represent a collection of objects (either files or other collections). These are the subtypes of DataCollection:
  - **CollOfDataCollection**: A collection of data collections.
  - **FileCollection**: A collection of Files

Computations (codes) are represented as workflow *components*. They can process several inputs and several outputs (each with its own unique id). A given input or output can take an entire data set. Components are organized in hierarchies of component types. The core definitions of the component ontology are:

- **ComponentType**: This is the top-level class of component types. A **Component** is an instance of this class and corresponds to an actual code that can be run.
- **ComponentCollection**: This represents a collection of components. It uses the property *hasComponentType* to specify the type of components in this collection.

ComponentCollection is used in nodes to indicate iterations over file collections.

Nodes in a workflow represent the component to be executed. A node in a workflow template can contain a single component or a component collection. A component collection is an intensional set of components, and will be expanded with concrete jobs when workflow instances are created. The definitions of nodes, and components are as follows:

- **Node**: Represents a node in the workflow. Uses a property *hasComponent* to specify the component that the node contains. Its range can be a any subclass of ComponentType or a ComponentCollection

A link in a workflow template carries data, and the type of data being carried must be consistent with the output data type of the origin node and the input data type of the destination node. Consequently, a link can carry single files or file collections. Links are defined as follows:

- **Link**: Represents a generic link in the workflow. It uses the properties *hasDestinationNode* and *hasOriginNode* to identify the destination and origin nodes respectively of the link. It also uses the properties *hasDestinationFileDescription* and *hasOriginFileDescription* to indicate the specific input/output for the components in the origin/destination nodes that this link connects. It has the following subclasses:
  - **InputLink**: These links do not have an origin node
  - **InOutLink**: These links must have an origin node and a destination node.
  - **OutputLink**: These links do not have a destination node.

Workflow templates are defined as including nodes that can be collections and links that can carry collections as well. This is done using the property *hasFile* of a link. Because we need to assert properties of these data collections, we need to represent them in the a-box as instances. Therefore, the data collections carried in the links are represented with *Skolem* instances, that is, instances that stand in for the actual data to be used in the instance. Properties and constraints can be asserted of these Skolem instances, which is important to validate the workflow. This requires that the entire workflow template is described with instances. Figure 3 shows an example of a workflow template, which corresponds to the following description:

```
<wfln:WorkflowTemplate rdf:ID="WT3">
     <wfln:hasLink rdf:resource="#InputFCSG_to_Cone"/>
     <wfln:hasLink rdf:resource="#InputFCSK_to_Cone"/>
     <wfln:hasLink rdf:resource="#Inout_from_Cone_to_Cmany"/>
     <wfln:hasLink rdf:resource="#InputFSY_to_Cmany"/>
     <wfln:hasNode rdf:resource="#Cone"/>
     <wfln:hasNode rdf:resource="#Cmany"/>
</wfln:WorkflowTemplate>
<wfln:InputLink rdf:ID="InputFCSG_to_Cone">
     <wfln:hasDestinationNode rdf:resource="#Cone"/>
     <wfln:hasDestinationFileDescription rdf:resource="&clib;#D1"/>
     <wfln:hasFile><FileCollection rdf:ID="#FCSG"/></wfln:hasFile>
</wfln:InputLink>
<wfln:InputLink rdf:ID="InputFCSK_to_Cone">
     <wfln:hasDestinationNode rdf:resource="#Cone"/>
     <wfln:hasDestinationFileDescription rdf:resource="&clib;#D2"/>
     <wfln:hasFile><FileCollection rdf:ID="#FCSK"/></wfln:hasFile>
</wfln:InputLink>
<wfln:InOutLink rdf:ID="Inout_from_Cone_to_Cmany">
     <wfln:hasOriginNode rdf:resource="#Cone"/>
     <wfln:hasDestinationNode rdf:resource="#Cmany"/>
     <wfln:hasOriginFileDescription rdf:resource="&clib;#D3"/>
    <wfln:hasDestinationFileDescription rdf:resource="&clib;#DC11"/>
     <wfln:hasFile><FileCollection rdf:ID="#FCSZ"/></wfln:hasFile>
</wfln:InOutLink>
<wfln:InputLink rdf:ID="InputFSY_to_Cmany">
     <wfln:hasDestinationNode rdf:resource="#Cmany"/>
     <wfln:hasDestinationFileDescription rdf:resource="#D12"/>
     <wfln:hasFile><File rdf:ID="#FSY"/></wfln:hasFile>
</wfln:InputLink>
<wfln:Node rdf:ID="Cmany">
     <wfln:hasComponent rdf:resource="&clib;Cmany"/>
</wfln:Node>
<wfln:Node rdf:ID="Cone">
     <wfln:hasComponent>
        <clns:ComponentCollection>
           <clns:hasComponentType rdf:resource="&clib;Cone"/>
        </clns:ComponentCollection>
     </wfln:hasComponent>
</wfln:Node>
```

Note that the number of elements in the collection is not specified since it is different for every instance to be created and depends on the size of the data set to be processed. Other constraints and properties of the set can be specified at the template level.

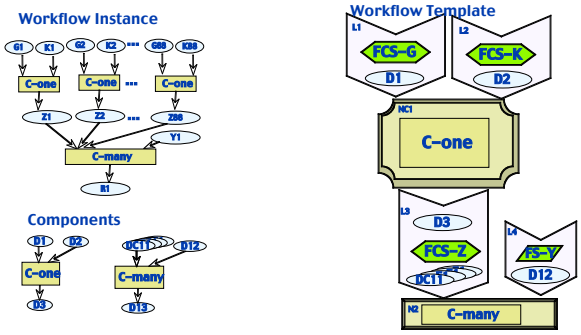


Fig. 3. An Example of a workflow template in Wings, shown on the right. A workflow instance from this template is shown on the left. Also shown is a sketch of the components.

So far, these simple manipulations of data collections have been sufficient. We are investigating additional constructs from parallel programming languages [28], similar to those available in Taverna [21].

Workflow instances are specified by binding the inputs of a workflow template to specific data sets. Here is an example of a workflow instance for the template above:

```
<wflns:WorkflowInstance rdf:ID="WI3">
            <wflns:hasWorkflowTemplate rdf:resource="&template;#WT3/>
            <wflns:hasBinding rdf:resource="#Binding0"/>
            <wflns:hasBinding rdf:resource="#Binding1"/>
            <wflns:hasBinding rdf:resource="#Binding2"/>
            <wflns:hasBinding rdf:resource="#Binding3"/>
</wflns:WorkflowTemplate>
<wflns:Binding rdf:ID="Binding0">
            <wflns:hasSkolemFile rdf:resource="&template;FCSG"/>
            <wflns:hasLibraryFile rdf:resource="&flib;CG1234567/>
</wflns:Binding>
<wflns:Binding rdf:ID="Binding1">
            <wflns:hasSkolemFile rdf:resource="&template;FCSK"/>
            <wflns:hasLibraryFile rdf:resource="&flib;CK1234567/>
</wflns:Binding>
<wflns:Binding rdf:ID="Binding2">
            <wflns:hasSkolemFile rdf:resource="&template;FCSZ"/>
            <wflns:hasLibraryFile rdf:resource="&flib;CZ1234567/>
</wflns:Binding>
<wflns:Binding rdf:ID="Binding3">
            <wflns:hasSkolemFile rdf:resource="&template;FSY"/>
            <wflns:hasLibraryFile rdf:resource="&flib; Y1234567/>
</wflns:Binding>
```

Wings can validate the creation of this instance by the user. Both collections provided as inputs for links L1 and L2 must have the same amount of elements. They must also comply with any constraints defined in the template.

Wings takes the descriptions of the input data and propagates them to create descriptions for all the data products of the workflow. For example, the collection in link L3 in the example is now known to have the same number of elements as the one in the link L1.

Note that this is an abbreviated specification of a workflow instance. Wings must then expand it to specify all the nodes that are to be executed. The algorithm for creating fully expanded workflow instances is shown in Figure 4.

```
CreateAbbreviatedWorkflowInstance
  Inputs: WorkflowTemplate tw, InputLinks Ilinks
  Output: Binding set for all Skolems in tw
   Assign ILinks to LinksToProcess.
   While LinksToProcess is not empty
     Remove one from LinksToProcess and assign it to L1.
     Let F1 be the file Skolem that is associated with the link L1.
   If F1 isn't already bound (check Bindings Map)
      If L1 is an Input link,
        Get file metadata from the user or file server
        Create new file in the file Library, FL1
        Bind File F1 to FL1, (assign F1 => FL1 in the Binding set)
     If L1 is an InOut Link or Output Link,
       Generate metadata automatically via propagation.
       Create new file in the Library, OFL1
       Bind File F1 to OFL1, (assign F1 => OFL1 in the Binding set)
   If all the links that L1 depends on have been processed
      For each link L2, that is dependent on L1, Add L2 to LinksToProcess
   Assign destination node of link L1 to N1.
   If all the inputs to N1 have been bound,
      Add all links with N1 as the origin node to LinksToProcess

CreateWorkflowInstance
  Inputs: WorkflowTemplate wt, Bindings bindings
  Output: WorkflowInstance wi
   For each Node N1 in the template wt,
      For each input file IF1 to N1,
         Get the Library File IBF1 from bindings, and replace IF1 as input to N1
      Do similarly for each output OF1
   Get the component C1, contained by N1.
      If C1 is a collection,
         Get number of items n in the collection from the number of items in input/output files for N1.
         Iterate n times, iteration i :
            Create job J1, with id = concatenate name of N1 with 'i'
            For each input IF1 to node N1,
               if IF1 is a Collection, then assign 'i-th' element of IF1 as an input to J1 (iteration)
              otherwise assign IF1 itself as an input to J1 (constant)
            Do similarly for each output OF1
     If C1 is not a collection,
       Create job J1, with id = name of N1.
        For each input IF1 to node N1,
           if IF1 is a Collection, then assign 'i-th' element of IF1 as an input to J1 (iteration)
           otherwise assign IF1 itself as an input to J1 (constant)
        Do similarly for each output OF1
```

Fig. 4. The algorithms to generate abbreviated and elaborated workflow instances in Wings.

Throughout the creation of the workflow instances, Wings propagates metadata information for all new data products. The metadata handling aspects of Wings are described in a separate paper [29] (available from URL provided in the citation), which describes this same algorithm in terms of how new metadata is generated for all new data products of the workflow.

Pegasus takes a very specific format for workflow instances. It is called a DAX (Directed Acyclic Graph in XML), and it is a directed acyclic graph of jobs where

each job consists of code and file names for the inputs and outputs of the job. It also takes specifications of which data must be registered, since some intermediate data may be of temporary utility only but others may be useful to the user. These are specified as defaults in the workflow template. Wings generates a workflow in DAX format, here is an excerpt of a very small one generated by Wings as an example:

```
<!-- part 1: list of all files used -->
  <filename file="file.f.a" link="input"/>
  <filename file="file.f.b1" link="inout"/>
  <filename file="file.f.b2" link="output"/>
<!-- part 2: definition of all jobs (at least one) -->
  <job id="ID000001" name="removeDups" version="1.0" level="3">
    <argument>-a top -T60  -i <filename file="file.f.a"/>  -o
<filename file="file.f.b1"/> </argument>
    <uses file="file.f.a" link="input" dontRegister="false"
dontTransfer="false"/>
    <uses file="file.f.b1" link="output" dontRegister="true"
dontTransfer="true" temporaryHint="true"/>
  </job>
  <job id="ID000002" name="countWords" version="1.0" level="2">
    <argument>-a left -T60  -i <filename file="file.f.b1"/> -o
<filename file="file.f.b2"/>  -p 0.5</argument>
    <uses file="file.f.b1" link="input" dontRegister="false"
dontTransfer="false" temporaryHint="true"/>
    <uses file="file.f.b2" link="output" dontRegister="true"
dontTransfer="true" temporaryHint="true"/>
  </job>
<!-- part 3: control-flow dependencies (empty for single jobs) -->
  <child ref="ID000002">
    <parent ref="ID000001"/>
  </child>
```

We have used Wings to create workflows for several applications. One kind of workflow is for rule pruning, as part of a statistical machine translation process. The template is shown in Figure 5. We created workflows of several dozen nodes that were then mapped by Pegasus and submitted for execution. A more recent application is seismic hazard analysis, where a workflow template of two dozen nodes was elaborated into a workflow instance of almost 8,000 nodes and sent to Pegasus for execution. This latter application of Wings is very complex and is described in detail in a separate submission [29] (available from URL provided in the citation).

The template in Figure 5 corresponds roughly to the structure of the example workflow shown in Figure 1-b.

5 Related Work

In our own prior work on the Composition Analysis Tool (CAT) [16] we addressed the validation of workflows and user assistance through semantic workflow representations, but we focused on smaller workflows of dozens of steps that did not handle large data sets.

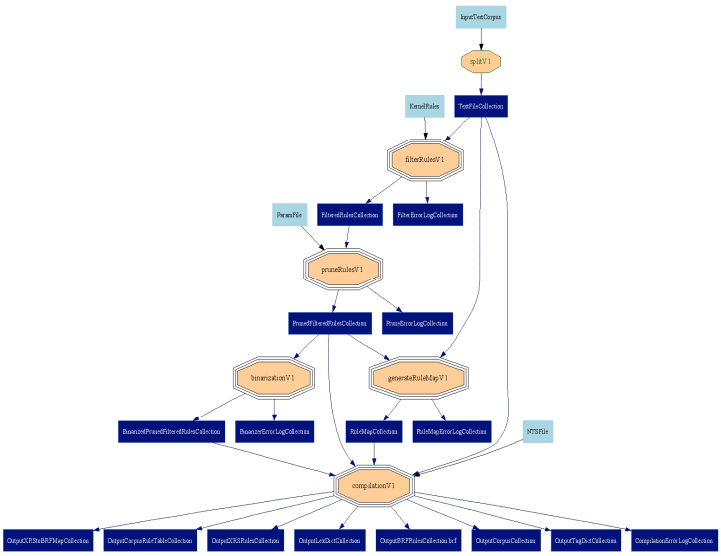


Fig. 5. A template created in Wings for pruning rules for a machine translation system.

Several approaches to creating workflows for scientific applications are widely used, prominently Taverna and Kepler [18,21]. Semantic descriptions are used to assist users in selecting workflow components and in ensuring that the final workflow is valid. However, their focus is on composing workflows from web services. In our work, we create workflows that are composed of computations and are executed over distributed grid computing environments [2,9]. The workflows that are created with these tools are complex but they handle data sets of smaller sizes. Kepler has some facilities to execute workflows in grid computing environments, where steps such as gridftp for data transfer are included in the workflow. We separate the execution concerns such as data movement and resource allocation from the higher-level specification of the workflow, leaving the former to Pegasus to handle.

Workflow languages, such as BPEL and FLOWS, support iteration and parallel constructs that can handle computation over data sets. However, they focus on compositions of web services rather than the workflows of computations that we need to support. In addition, these languages do not support the representation of semantic constraints and other information about metadata. Workflow editors for these languages help users create workflows through graphical manipulations, but do not include facilities for validating workflows according to semantic constraints. The Virtual Data Language (VDL) [26] allows users to create workflows that are composed of computations over large data sets. Users specify workflows portions that are processed by Chimera to assemble a complete abstract workflow. The complete file names must be spelled out in the initial specification. In recent work, data sets are represented explicitly and handled by indicating iteration constructs, in that sense VDL is moving closer to providing a scripting language designed to support workflow specification. Semantic descriptions of data and other constraints imposed by the workflow structure are not supported in the language.

A common way for users to orchestrate complex computations over large data sets is to create workflows through portals [3]. These portals have hardcoded workflows and default settings, and ask users to provide scenario-specific data that are then transformed programmatically into the sequences of computations to be performed.

6 Conclusions

We have described a new approach to creating and validating very large scientific workflows that handle data sets through computational steps that are executed in distributed grid environments. Our approach uses semantic descriptions of workflow templates and workflow instances where all their constituents are semantic objects that are described with properties and workflow level constraints. Once a workflow template is created and validated by an experienced user, it is easy for more junior scientists to create sophisticated analyses simply by specifying input data for pre-defined templates. The system ensures that the input data specified is appropriate given the definitions in the workflow template, and automatically generates a workflow instance that can be mapped to execution resources. We have implemented this approach in the Wings system, and is fully integrated with the Pegasus workflow mapping system. Augmenting workflows with semantic descriptions also enables searches of previous instances or templates in cases where a "similar" analysis is sought. A scientist may want to find out if someone else has come across a particular problem or used a particular methodology. The explicit representations of the workflow templates and instances support result reproducibility. Templates provide a means of systematically and diligently describing the high-level analytical steps involved. Workflow instances created from those templates are valid since they follow established methodology (as described in the template) and the data complies with the constraints expressed in the workflow.

Acknowledgments. We gratefully acknowledge our many collaborators, in particular from the Southern California Earthquake Center (SCEC) and the Machine Translation research group at USC's Information Sciences Institute for sharing with us their challenging workflows. We would also like to thank Gaurang Mehta and Mei Hsu for their assistance in running Wings workflows in Pegasus. This research was funded by a grant from the National Science Foundation EAR-0122464 and by internal research funds from the Information Sciences Institute.

References

1. J. Annis, Y, Zhao, J. Voeckler, M. Wilde, S. Kent, and I. Foster. Applying Chimera Virtual Data Concepts to Cluster Finding in the Sloan Sky Survey. Proceedings of Supercomputing 2002, Baltimore, MD, November, 2002.
2. Berman, F., Hey, A. J.G., and Fox, G. "Grid Computing: Making The Global Infrastructure a Reality" John Wiley & Sons, 2003.
3. BIRN: Biomedical Informatics Research Network, http://www.nbirn.net/.
4. Blythe, J., Deelman, E., Gil, Y., Kesselman, C. "Transparent Grid Computing: A Knowledge-Based Approach", Proceedings of the 15th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI), August 12-14, 2003, Acapulco, Mexico.
5. Blythe, J., Deelman, E., Gil, Y., Kesselman, C., Agarwal, A., Mehta, G., Vahi, K. "The Role of Planning in Grid Computing", Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS), June 9-13, 2003, Trento, Italy.
6. Deelman, E., Blythe, J., Deelman, E., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Blackburn, K., Lazzarini, A., Arbree, A., Cavanaugh, R., and Koranda, S. " Mapping Abstract Workflows onto Grid Environments", Journal of Grid Computing, Vol. 1, No. 1, 2003.

7. Ewa Deelman, Jim Blythe, Yolanda Gil, Carl Kesselman, Scott Koranda, Albert Lazzarini, Gaurang Mehta, Maria Alessandra Papa, and Karan Vahi. "Pegasus and the Pulsar Search: From Metadata to Execution on the Grid". PPAM Applications Grid Workshop (AGW), Czestochowa, Poland, 2003.
8. Ewa Deelman, Jim Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei-Hui Su, Karan Vahi, and Miron Livny. "Pegasus: Mapping Scientific Workflows onto the Grid". Across Grids Conference, Nicosia, Cyprus, 2004.
9. Foster, I., and Kesselman, C. "The Grid: Blueprint for a New Computing Infrastructure", 2nd Edition, Morgan Kaufmann, 2004.
10. Gannon, D., Deelman, E., Taylor, I., and Shields, M. (Eds) "Workflows in e-Science", Springer Verlag, forthcoming.
11. Gil, Y. "Workflow Composition: Semantic Representations for Flexible Automation", in "Workflows for e-Science", Deelman, E., Gannon, D. Shields, M., and Taylor, I. (Eds), Springer Verlag, 2006.
12. Gil, Y. and Ratnakar, V. "Multi-Agent Systems and Grid Services: Towards Robust Continuous Distributed Problem Solving". Internal Project Report, October 2003.
13. Yolanda Gil, Ewa Deelman, Jim Blythe, Carl Kesselman, and Hongsuda Tangmurarunkit. "Artificial Intelligence and Grids: Workflow Planning and Beyond", IEEE Intelligent Systems, January 2004.
14. Yolanda Gil, Varun Ratnakar, and Ewa Deelman. "Augmenting Metadata Catalogs with Semantic Representations", Short paper at the Fourth International Semantic Web Conference (ISWC-05), Galway, Ireland, November 7-10, 2005.
15. Yolanda Gil, Varun Ratnakar, and Ewa Deelman. "Virtual Metadata Catalogs: Augmenting Metadata Catalogs with Semantic Representations", International Provenance and Annotation Workshop (IPAW'06), Chicago, IL, May 3-5, 2006.
16. Jihie Kim, Marc Spraragen, and Yolanda Gil. "An Intelligent Assistant for Interactive Workflow Composition", In proceedings of the 2004 International Conference on Intelligent User Interfaces (IUI), Madeira Islands, Portugal, January 2004.
17. K. Knight and D. Marcu. Machine Translation in the Year 2004. Proceedings of the 2005 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2005.
18. B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, Y. Zhao, Scientific Workflow Management and the Kepler System. In Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows, to appear, 2005.
19. Ludaescher B. and Goble, C. Special Issue on Scientific Workflows, ACM SIGMOD Record, September 2005.
20. P. Maechling, H. Chalupsky, M. Dougherty, E. Deelman, Y. Gil, S. Gullapalli, V. Gupta, C. Kesselman, J. Kim, G. Mehta, B. Mendenhall, T. Russ, G. Singh, M. Spraragen, G. Staples, and K. Vahi. "Simplifying Construction of Complex Workflows for Non-Expert Users of the Southern California Earthquake Center Community Modeling Environment". In ACM SIGMOD Record, special issue on Scientific Workflows, 2005.
21. T. Oinn, M. Greenwood, M. Addis, M. Nedim Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat and C. Wroe. Taverna: Lessons in creating a workflow environment for the life sciences accepted for publication in Concurrency and Computation: Practice and Experience Grid Workflow Special Issue
22. Open Science Grid, www.osg.org.
23. G. Singh, S. Bharathi, A. Chervenak, E. Deelman, C. Kesselman, M. Manohar, S. Patil, L. Pearlman. A Metadata Catalog Service for Data Intensive Applications. SuperComputing, 2003.
24. TeraGrid, www.teragrid.org.
25. Tuchinda, T., Thakkar, S., Gil, Y., and Deelman, E. "Artemis: Integrating Scientific Data on the Grid", Proceedings of the 16th Conference on Innovative Applications of Artificial Intelligence (IAAI), San Jose, CA, July 25-29, 2004.
26. Y. Zhao, J. Dobson, I. Foster, L. Moreau and M. Wilde. A Notation and System for Expressing and Executing Cleanly Typed Workflows on Messy Scientific Data. ACM SIGMOD Record, special issue on Scientific Workflows, 2005.
27. M. Wieczorek, R. Prodan and T. Fahringer. Scheduling of Scientific Workflows in the ASKALON Grid Environment. ACM SIGMOD Record, special issue on Scientific Workflows, 2005.
28. E. Deelman, M. Hall, Y. Gil, K. Lerman, and J. Saltz, "A Systematic Approach to Composing and Optimizing Application Workflows," In Proceedings of the 2005 Workshop on Patterns in High Performance Computing, May, 2005.
29. Kim, J., Gil, Y., and Ratnakar, V. Semantic Metadata Generation for Large Scientific Workflows. Submitted to the International Semantic WebConference, 2006, available at http://www.isi.edu/ikcap/scec/papers/Wings-metadata-reasoning.pdf.