Proceedings of the Fifth IEEE International Conference on e-Science (e-Science 2009), Oxford, UK, December 9-11, 2009.

Expressive Reusable Workflow Templates

Yolanda Gil, Paul Groth, Varun Ratnakar, Christian Fritz
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA, 90292, USA
{gil, pgroth, varunr, fritz}@isi.edu

Abstract

Workflow systems can manage complex scientific applications with distributed data processing. Although some workflow systems can represent collections of data with very compact abstractions and manage their execution efficiently, there are no approaches to date to manage collections of application components required to express some scientific applications. We present an approach to handle collections of components and data alike in expressive workflow templates whose basic structure is reusable. We also present an algorithm that can elaborate abstract compact workflow templates into execution-ready workflows that enumerate all computations to be carried out. We implemented the proposed approach in the Wings workflow system. Our work is motivated by real-world complex scientific applications that require handling of nested collections of both components and data.

1. Introduction

Scientists often deal with collections of data, whether it is multiple overlaying images produced by fMRI scanners or thousands of gene sequences generated using high throughput sequencing. To deal with such large and complex collections of data, scientist have turned towards workflow technology. Computational experiments can be modeled as workflows, which are declarative representations of the dataflow between software components. Thus, sophisticated software packages can be weaved together in order to express a computational experiment. Once an experiment is represented as a workflow, workflow systems can be used to execute computational experiments on a large scale [3], optimize performance [15] and track the provenance of experimental outputs [7]. One important outcome of representing experiments as workflows is the ability for scientists to easily share and reuse experiments [11].

However, the software components within a workflow are in many cases not designed to process more than one data set at a time. Consider, for example, a bioinformatician, testing whether a particular gene expression predicts a given phenotype in an organism using a k-nearest neighbor classifier. This classifier typically only classifies one data set at a time, but imagine that the bioinformatician wants to test a newly trained classifier on many test data sets in order to have evidence of the classifier's efficacy. Or, to find the classifier that produces the best results, the bioinfomatician may want to train and test a collection of alternative algorithms simultaneously. To support this sort of application using available software components, a workflow system needs to be able to represent, reason about and process not only collections of data but also collections of components.

In order to further enable the sharing and reuse of computational experiments, workflows need to be able to be easily adapted both to new or similar data sets and the availability of new analysis components. For example, if new test data sets or classifiers are available for use by the aforementioned bioinformatician, the workflow should be easily (and perhaps automatically) adapted to them. Thus, the workflow system should make it easy to reuse the basic dataflow structure of an experiment at an abstract level, and dynamically incorporate new data sets and components.

In this paper, we present a new approach to workflow representation and generation that addresses collections of data and components. While some workflow systems have treated collections [5;14;9], our approach to collections differs in that it 1) handles collections of components in addition to collections of data, and 2) automatically adapts the initial workflow template to new collections of data sets and components. Our approach is implemented as an extension to the Wings workflow system [4;7].

The paper starts with motivating examples that lead to requirements to handle collections. We then describe the representations of workflow templates that we have developed to support those requirements. We also present the algorithm that uses those representations to map the abstract workflow templates into execution-ready workflows that enumerate all needed computations. Finally, we describe how a number of workflows taken from the literature can be expressed with the proposed approach.

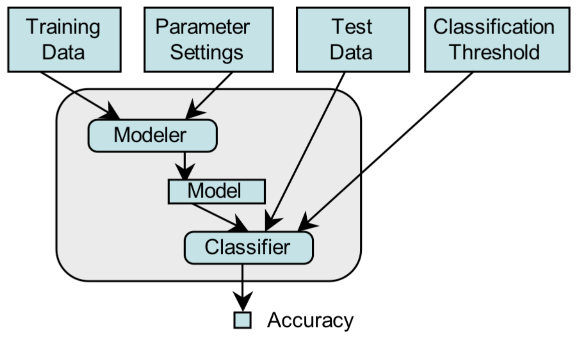
2. Requirements on Workflow Templates

Workflow templates represent abstractions over the actual workflow computations to be submitted to execution. For example, if an executable workflow is to process in parallel two datasets of n1 and n2 elements each we could imagine a workflow template that expresses that the computation is to be executed with the cross-product of the two sets or with their pair-wise combination. We refer to each computation as a workflow component, which essentially represents some executable code whose execution and dataflow are managed by the workflow system[1].

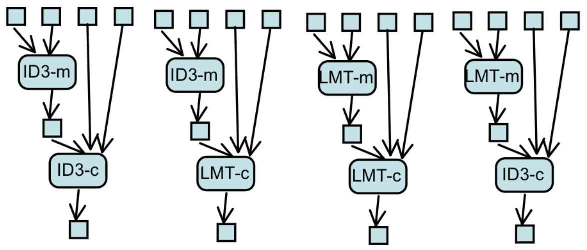
For illustration purposes, we use examples of data mining algorithms and workflows since they are common in science, from bioinformatics to astrophysics to chemistry. We look at classification tasks, where a model is used to classify a set of test data. There are several widely-used approaches to building a model from a set of training data, such as decision trees (DT) and k-nearest neighbor (KNN). Within each approach several algorithms are possible. For example, decision tree algorithms include a classic divide and conquer algorithm (ID3) and a logistic model tree builder (LMT). This results in a hierarchy of algorithm classes. The class Modeler includes the subclass DecisionTree-Modeler, which includes ID3-m and LMT-m as possible workflow components. The class Classifier includes the subclass DecisionTree-Classifier, which includes ID3-c and LMT-c as possible workflow components. KNN-m is a Modeler that is not in the DecisionTree-Modeler class, and KNN-c is a Classifier that is not in the DecisionTree-Classifier class. We will use this very simple class hierarchy of workflow components in our examples.

Figures 1 and 2 show several illustrative examples of desirable workflow abstractions. Workflow diagram MTC (Model Then Classifier) at the top of Figure 1 shows a dataflow structure where a modeler is trained with a training dataset using some parameter settings to produce a model, then a classifier uses the model to classify a test dataset and produce some accuracy measurement about the model. Workflow diagram OA (Obtain Accuracy) at the top of Figure 2 shows a

MTC (ModelThenClassify): Workflow for modeling and classifying to produce an accuracy estimate.



MTC-DT: User desires a set of workflows, one per possible combination of decision tree algorithms.



MTC-PS: User desires a set of workflows to find the accuracy of a given algorithm (e.g. KNN) with a series of parameter settings.

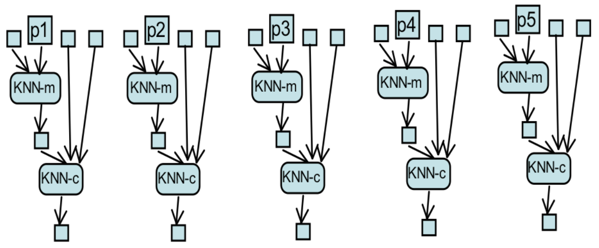


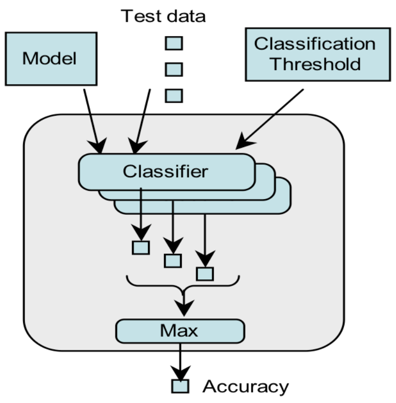
Figure 1. Workflows for modeling and classifying datasets using different selections of algorithm sets.

dataflow structure where the maximum accuracy of a model is obtained for a set of test data. MTC and OA could each be a reasonable reusable workflow template. However, consider the following variants of those templates that a scientist may want to express. As we will show later, the variants shown here are simplifications of real cases of scientific workflows.

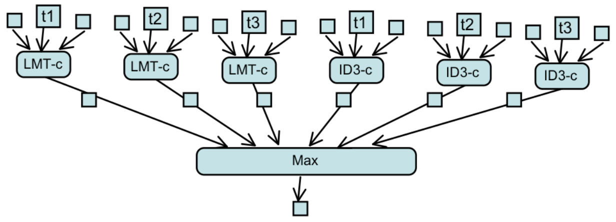
A variant of MTC is to use that basic template to generate a set of workflows, one for each possible combination of modeler and classifier algorithms (for example, algorithms based on decision trees). We will

[1] The discussion here focuses on command-line applications but can also be adapted for use with web services as components.

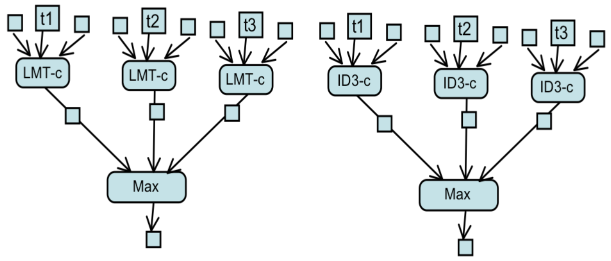
OA (ObtainAccuracy): Workflow for estimating the maximum accuracy of a model for a set of test data.



OA-DT: User desires a workflow for estimating the maximum accuracy of a set of algorithms.



OA-DTW: User desires a workflow for estimating the maximum accuracy of each of a set of algorithms.



OA-M: User desires a workflow to find the maximum accuracy of an algorithm using a set of models

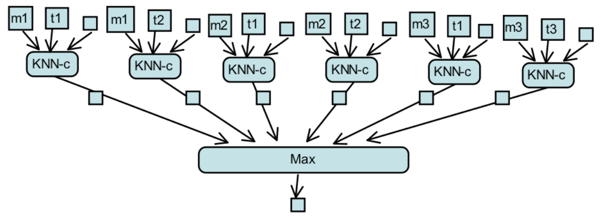


Figure 2. Workflows for measuring the accuracy of different algorithm sets.

refer to this variant as MTC-DT. A second variant of MTC is to use that same basic template to generate a set of workflows, one for each of a series of parameter settings for a k-nearest neighbor based modeler and classifier. We will refer to this variant as MTC-PS. For both these variants, the user would obtain several workflows and explore which one serves their needs best.

To represent these variants of the same MTC workflow template, we need to express:

- Iterations of a workflow over a set of data (e.g., the set of parameter settings for MTC-PS).
- Iterations of a workflow over a set of algorithms (e.g., the set of decision tree algorithms in MTC-DT).
- Algorithm types that are valid for the particular user request (e.g., the class of decision tree algorithms in MTC-DT).
- A specific algorithm to be used (e.g., KNN in MTC-PS).

Let us consider some variants on the OA template. A scientist may want to obtain the maximum accuracy of a class of algorithms (e.g., decision tree algorithms). We refer to this variant as OA-DT. A second variant, OA-DTW, would obtain the maximum accuracy for each of the algorithms in a set separately. A third variant would use the same basic OA template but obtaining the maximum accuracy for one algorithm (for example, a k-nearest neighbor) using a set of models and a set of training data. We refer to this variant as OA-M. Note that for OA-DTW separate workflows are created as in the variants of the MTC template from Figure 1, but for the other variants the result is a single workflow, where the results of the sets are accumulated by the Max component. Representing these variants poses some additional requirements:

- Creating a set of components each for an algorithm in a given set (e.g., a component for each decision tree algorithm in the case of OA-DT).
- Creating a set of components each to process a data element in a given data set (e.g., a component for each model and each test pair in the case of OA-M).

The next section describes how these requirements can be addressed with a representation of workflow templates that can specify the variants above.

3. Expressive Representations of Workflow Templates

Our approach to representing workflow templates is based on several key ideas:

1. We use workflow variables to represent datasets as well as workflow components.
2. We allow semantic constraints to be associated with workflow variables, to express whether they represent individuals or sets, what their types are, other properties such as the sizes of sets, and the bindings to individuals or sets.

3. We use special constructs to specify how sets of components or sets of data are to be handled by the workflow system when elaborating the template.

We have also developed an algorithm that uses these representations to map workflow templates into workflow instances that can be submitted to an execution engine. That is, with our workflow template language we need to be able to express templates to reflect the kinds of workflow variants shown in Figures 1 and 2, and with our mapping algorithm we need to be able to generate workflow instances that have elaborated each specific computation to be executed as shown in the variants discussed earlier. We describe the algorithm for elaborating workflow templates and mapping them to workflow instances in Section 4.

The rest of this section describes how workflow templates are represented. In our examples, any assertions about a workflow will be given as triples. In our implementation, workflows are represented using the W3C's Web Ontology Language (OWL) standard[2], as are components and datasets. We use N3-like notation[3] in our examples. We believe that the same underlying model could be incorporated in alternative representation frameworks where object types and properties can be expressed.

3.1 Structure of Workflow Templates

A workflow template includes a specification of the basic dataflow structure of the workflow as a graph of nodes and links. The structure does not contain any repetitions of components or datasets, rather any repetition is compactly represented by a variable constrained to take a set. Specifically, each individual dataset or dataset collection is assigned a unique variable, and each individual component or repetition of components is assigned a unique variable. Component variables are assigned to nodes, and data variables are assigned to links. Links express dataflow across components as their origin and destination nodes, unless they are input or output of the workflow in which case they are missing their origin or destination, respectively.

For example, to express the Modeler node in the MTC workflow and its surrounding links we would assert the following:

```
Modeler-Node has-var Modeler-Var
Modeler-Var type Modeler
Classifier-Node has-var Classifier-Var
Classifier-Var type Classifier
Link1 type InputLink
Link1 has-var TrainingData-Var
Link1 has-destination Modeler-Node
Link2 type InputLink
Link2 has-var ParameterSettings-Var
Link2 has-destination Modeler-Node
Link3 has-var Model-Var
Model-Var type Model
Link3 has-origin Modeler-Node
Link3 has-destination Classifier-Node
```

Nodes can also be assigned subworkflows. For example a workflow could have the MTC subworkflow in a node:

```
GetAccuracy-Node has-var GetAccuracy-Var
GetAccuracy-Var type MTC
```

We will mention later on how we express the connections between dataflow links and the arguments of components.

3.2. Semantic Constraints on Data and Component Collections

We express semantic constraints on workflow data variables. One use of semantic constraints, as seen before, is to specify types. For example in the MTC-DT workflow we can express:

```
Modeler-Var type DecisionTreeModeler
```

This means that any Modeler algorithm that is in the subclass DT (decision trees) can be assigned to that variable.

Another use of semantic constraints is to express bindings, i.e., assignments of specific datasets or a specific algorithms to the variables. For example in the MTC-PS variant:

```
Modeler-Var has-binding KNN-m
```

This means that the variable has been assigned the modeler algorithm for k-nearest neighbor.

We also use semantic constraints to represent collections of datasets, as well as constraints on those collections. Collections have dimensions that allow nesting, and have a size along each dimension. For example, in the OA-M variant:

```
Model-Var has-dim 1
Model-Var has-binding m1 m2 m3
```

That is, the input to the workflow will be a set of models.

We handle sets of parameters in the same way that we handle sets of data.

We can also enumerate the elements of the set by associating constraints to the variable. For example, OA-DTW could be expressed by stating the specific algorithms are to be used as modelers:

2 http://www.w3.org/2004/OWL/

3 http://www.w3.org/DesignIssues/Notation3.html

```
Classifier-Var has-dim 1
Classifier-Var has-binding ID3-c LMT-c
```

This way, we can express both intensional or extensional sets associated with a workflow variable.

Sets of data can be represented similarly. Multi-dimensional sets can also be represented. For example, to represent that a 3D image can be broken into 2D layers each layer containing tiles we would state:

```
Image-Var has-dim 2
Image-Var type Tile
```

Semantic constraints can also be expressed among variables in a workflow and any subworkflows assigned to its nodes. In our framework, we use namespaces to refer to subworkflow variables.

3.3. Component Collections and Mappings

We need to express what behavior we expect from the system when elaborating a workflow template that has sets of components or sets of datasets. In elaborating the template, each node and link containing sets has to be *mapped* into a set of nodes and links. However, there are different ways to do those mappings, and we have special constructs for each kind.

There are two distinct cases of nodes and links that need to be mapped: 1) *component mappings* or *c-mappings*, when a node in a workflow template has a component variable containing a set of components, and 2) *data mappings* or *d-mappings*, when a node is the destination of a link whose data variable has a higher dimensionality than the inputs of the component of the node's component variable.

There are two constructs to handle c-mappings. The *WC ("a Workflow per Component")* construct expresses that alternative workflows need to be created for each element of the set of components. To achieve the MTC-DT variant we would state in the template:

```
Modeler-Var type DecisionTreeModeler
Modeler-Node has-mapping WC
Classifier-Var type DecisionTreeClassifier
Classifier-Node has-mapping WC
```

So each node in MTC would be spawning a new workflow per algorithm. The result is several workflows, each with a possible combination of decision tree algorithms.

The *BC ("a Branch per Component")* is another construct for c-mappings. It expresses that alternative branches need to be created for each element in the set of components. The OA-DT variant can be achieved by stating in the template:

```
Classifier-Node has-mapping BC
```

It is important to note that this BC construct leaves all the branches within the same workflow, while the WC construct results in the creation of new workflows.

There are two analogous constructs to handle d-mappings. The *WD ("a Workflow per Dataset")* construct indicates that alternative workflows need to be created for each extra dimension and element of the set of the data variable. The MTC-PS variant can be expressed using this construct as follows:

```
TestData-Var has-binding p1 p2 p3 p4 p5
Link3 has-var ParameterSettings-Var
Link3 has-destination Modeler-Node
Modeler-Node has-var Modeler-Var
Modeler-Var has-binding KNN-m
Modeler-Node has-mapping WD
```

Five separate workflows would result in this case.

The *BD ("a Branch per Dataset")* construct is also used for d-mappings. It expresses that alternative branches need to be created for each extra dimension and element of the set of the data variable. The OA-M variant can be created by stating:

```
Model-Var has-binding m1 m2 m
Link3 has-var ModelData-Var
Link3 has-destination Classifier-Node
Classifier-Node has-var Classifier-Var
Classifier-Var has-binding KNN
Classifier-Node has-mapping BD
```

The result would be four branches in the same workflow, each corresponding to one of the models.

Section 4 describes how these constructs are used within our algorithm to elaborate workflow templates.

3.4. Arguments to Subworkflows and Components

Each component has roles, which are unique identifiers for its arguments. Similarly, each subworkflow has roles. Each node has ports, which are unique identifiers for the arguments of its component's roles. We support cross-product and pairwise combinations of datasets coming to ports in a node into the roles of the individual components.

4. Workflow Elaboration Algorithm

Recall that the purpose of the workflow system is to elaborate template workflows into executable ones. For that, the system needs to interpret the constructs that are used to describe the mapping of sets that are used in specifying the workflow template. We present here a workflow elaboration algorithm that builds on the Wings workflow generation algorithm [4].

The algorithm elaborates workflows in two major phases: (1) component selection and (2) data projection. In Phase 1, starting from the end data products in the workflow, the workflow is traversed backwards. In this traversal, when nodes are not bound to specific components yet, the component catalog (which stores classes of components) is queried to obtain a (possibly empty) set of appropriate components to which the node's component variable can be bound. This selection is made based on the constraints that are specified in the workflow template and additional constraints that are propagated backwards from the data products of the workflow. During the back-propagation, the component catalog is queried to obtain constraints on the inputs of specific components that are necessary for the component to produce the desired output. If a component is not able to produce the desired output under any circumstances, it is ruled out.

During this first phase, a query to the component catalog may return more than one eligible component. This is where the mapping constructs WC and BC come into play. If the node is marked as a WC, the algorithm produces separate workflows for each eligible component. In case of a BC, the algorithm creates separate branches within the same workflow instead, i.e., it creates several copies of the node, one for each eligible component returned by the component catalog. At the end of Phase 1, the constraints have been propagated to the workflow inputs.

In Phase 2, the constraints on the data inputs provided by the user are propagated forward through the workflow(s). As we mentioned earlier, a variable for input data could be bound to a set of several dimensions. As with the component selection, the choice of mapping constructs, WD or BD, is used to decide how to proceed when the dimensionality of the input data to a node's port does not match the dimensionality of its component's corresponding input role. In such a case, when WD is used to mark the port of the node, a separate workflow is created for each data item in the set. In case of BD, separate branches within the same workflow are created, i.e., copies of the nodes are created for each input element. Given the constraints unveiled by Phase 1, though, some of these elements may be rejected as invalid. It is important to note that certain components may produce the same number of outputs, independent from the number of inputs. This is the case, for instance, for the "Max" in the OA workflow template, which aggregates its input data sets into a single output element. This is the reason why it is not the case that an increase of the dimensionality or size of inputs always results in an equal increase in dimensionality of all subsequent data products and nodes.

The workflows variants shown at the bottom of Figures 1 and 2 are results of running this algorithm given different mapping constructs (WC, BC, WD, BD) and data input dimensions. These resulting workflows can be converted to a format appropriate for the execution engine of choice, in our case Pegasus [3].

Table 1 shows how the workflows in Figure 1 and 2 would be achieved using our approach. Note that it is possible to associate more than one of these constructs within the same workflow.

Workflow Variant	**Collection Handling**
MTC-DT	WC on modeler variable WC on classifier variable
MTC-PS	WD on parameter settings variable
OA-DT	BC on classifier variable BD on test data variable
OA-DTW	WC on classifier node variable BD on test data variable
OA-M	BD on test data variable BD on model variable

Table 1. Representation of Example Workflows.

5. Representing Scientific Workflows

This section shows how our approach is used to create scientific workflows described in the literature. The workflows included here were selected to illustrate the variety of capabilities of our approach.

[8] describes an application where we used Wings to manage workflows for biomedical imaging. A workflow template from that work is depicted in Figure 3. We exploited the management of datasets as well as reasoning about semantic constraints on data collections in order to anticipate the amount of computations and data products to be expected from the workflow. Using this information, a tradeoff module was able to make informed decisions about how many resources to allocate to the workflow and how to set parameters for the individual algorithms based on quality/performance tradeoffs. In our case, the workflow was then submitted to the Pegasus workflow mapping and execution engine for execution over distributed resources.

[10] describes GenePattern, a system that can help users manage abstract protocols for genomic analysis that can be specialized into executable pipelines (workflows). Each component in a protocol may correspond to many executable modules, similar in nature to our discussion above regarding component classes and subclasses. In their system, the protocols are specialized by the user. Figure 4 illustrates with a diagram a workflow template of one of the abstract protocols available in their system. Using the proposed

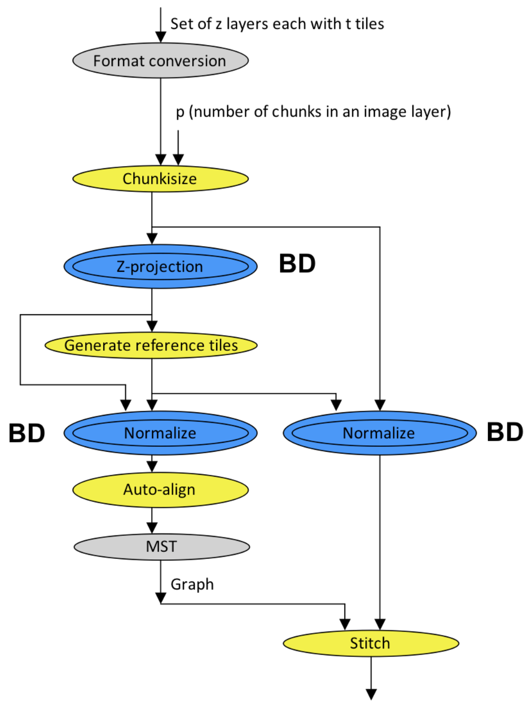


Figure 3. Sketch of a workflow template for biomedical imaging used in [9].

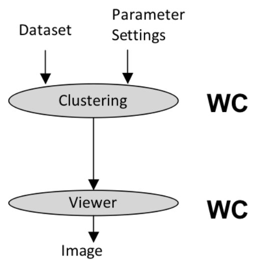


Figure 4. Sketch of a workflow template to represent an abstract protocol for genomic analysis from [11].

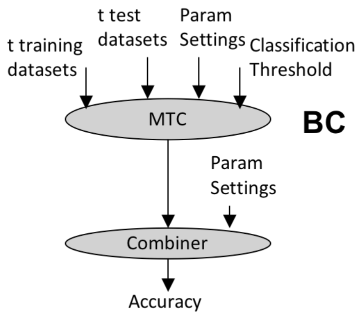


Figure 5. Sketch of a workflow for protein secondary structure prediction from [17].

approach in Wings, we are able to specialize the protocols into executable pipelines automatically.

[16] describes an approach to protein secondary structure prediction that uses several algorithms and a combination function. Figure 5 illustrates the abstract template that would correspond to the workflow representing that approach.

These real-world workflows illustrate the potential of our approach. If scientists had at their disposal expressive reusable workflow templates and workflow systems that could interpret, specialize, and execute them, they would have a very powerful experimental apparatus. Data analysis workflows are becoming increasingly more complex, due to the use of sophisticated statistical techniques and the increasing diversity of algorithms available to them. For example, the workflows shown in Section 2 are taken from work that we are doing to represent nested cross-validation approaches and optimization techniques for parameter selection that are frequently used in genomics and other sciences.

6. Related Work

Simplified approaches to dealing with collections of data have a long tradition in computer science. Sipelstein and Blelloch provide an extensive review of collection-oriented programming languages [12]. While our focus is on workflow systems that deal with collections, it is important to note that much of this work adopts the approaches of programming languages and in particular the use of higher-order functions such as "map" that apply a function to all elements in a list.

Hidders et al. combine the Nested Relational Calculus [2] and Petri-nets in a workflow language called DFL to be able to deal with operations on collections within in the context of data flow centric workflows [5]. The Nested Relational Calculus is a super set of comprehension syntax, commonly used in collection oriented languages [1]. Our approach differs from DFL in that Wings automatically determines how a collection should be decomposed and recombined for a particular component. In DFL, the user must specify how this is done using explicit nest and unnest operations.

Unlike DFL, Taverna is able to automatically adapt a collection to a component [14]. For example, when a component accepts a single data item as input and is provided a list, Taverna performs an implicit iteration using the component over the input data set. If there are multiple inputs to the same component, then Taverna performs either a dot or Cartesian product on the input data sets and then invokes the implicit iteration over the resulting list. For each workflow component, the choice of dot or cartesian product is specified by the user [13]. Hidders et al. provide an alternative formal description of the semantics of a Taverna workflow that treats collections but also deals with failures and side effects [6]. Unlike Taverna, our approach is to reason about collections prior to execution. That is, a user or execution system can determine how a data collection will impact a workflow prior to execution. Additionally, Wings handles not only data collections but also collections of components. Thus, a user can more easily express an experiment in which multiple components of the same type can be applied to a collection of data. Interestingly, [14] hypothesizes the ability to generate an executable version of a workflow from a high-level description. Wings realizes this hypothesized functionality.

Kepler provides another approach to handling collections of data. It specifies a particular data structure for collections, namely a flat list of data elements separated by tokens [9]. Once a collection is transformed into Kepler's format, Kepler models a workflow as an "assembly line", where the collection is given to each component in the order defined by the workflow. Each component is then responsible for selecting the part of the collection that it needs to operate on. Additionally, the component is responsible for adding to the collection in a compatible manner. This approach has several advantages including the ability to incrementally modify a collection and the elimination of complex control structures. However, unlike our approach, it requires that both the data sets and components are modified to conform with the requirements of Kepler's approach.

7. Conclusions

Scientific applications often require complex handling of collections of both components and data. This paper builds upon our prior work in the Wings workflow system and demonstrates an additional layer of adaptability that allows a workflow template to be applied and reused with collections that change its structure. A novel contribution is the handling of collections of components.

Acknowledgements

We gratefully acknowledge funding from the National Science Foundation under grant CCF-0725332. We thank Jihie Kim and Ewa Deelman for many fruitful discussions. We also thank Vijay Kumar, Michael Reich, and other collaborators who inspired the research described in this paper.

References

[1] P. Buneman, L. Libkin, D. Suciu, V. Tannen, and L. Wong. "Comprehension syntax". SIGMOD Rec., 23(1):87–96, 1994.

[2] P. Buneman, S. Naqvi, V. Tannen, and L. Wong. "Principles of programming with complex objects and collection types". Theoretical Computer Science, 149(1):3–48, 1995.

[3] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, J. Kim, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, D. S. Katz. "Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems". Scientific Programming, Vol. 13, No. 3, 2005.

[4] Y. Gil, V. Ratnakar, E. Deelman, G. Mehta, and J. Kim. "Wings for Pegasus: Creating large-scale scientific applications using semantic representations of computational workflows". In Proceedings of 19th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI), Vancouver, British Columbia, Canada, July 22-26 2007.

[5] J. Hidders, N. Kwasnikowska, J. Sroka, J. Tyszkiewicz, and J. V. den Bussche. "DFL: A dataflow language based on Petri nets and nested relational calculus". Information Systems, 33(3):261 – 284, 2008.

[6] J. Hidders and J. Sroka. "Towards a calculus for collection-oriented scientific workflows with side effects". In On the Move to Meaningful Internet Systems 2008: volume 33, pages 261–284, Oxford, UK, 2008. Elsevier Science Ltd.

[7] J. Kim, E. Deelman, Y. Gil, G. Mehta, and V. Ratnakar. "Provenance Trails in the Wings/Pegasus Workflow System," Concurrency and Computation: Practice and Experience, Vol 20, Issue 5, April 2008.

[8] V.S. Kumar, P. Sadayappan, G. Mehta, K. Vahi, E. Deelman, V. Ratnakar, J. Kim, Y. Gil, M. Hall, T. Kurc, and J. Saltz. "An Integrated Framework for Parameter-based Optimization of Scientific Workflows," Proceedings of the 18th International Symposium on High Performance Distributed Computing (HPDC), Munich, Germany, June 11-13, 2009.

[9] T. Mcphillips, S. Bowers, and B. Ludaescher. "Collection oriented scientific workflows for integrating and analyzing biological data". In Data Integration in the Life Sciences, pages 248–263. 2006.

[10] M. Reich, T. Liefeld, J. Gould, J. Lerner, P. Tamayo, and J. P. Mesirov (2006) "GenePattern 2.0". Nature Genetics 38 no. 5 (2006): pp500-501 doi:10.1038/ng0506-500.

[11] D. D. Roure, C. Goble, and R. Stevens. "The design and realisation of the virtual research environment for social sharing of workflows." Future Generation Computer Systems, 25(5):561 – 567, 2009.

[12] J. M. Sipelstein and G. E. Blelloch. "Collection-Oriented Languages". Proceedings of the IEEE, 79(4):504–523, 1991.

[13] W. Tan, P. Missier, R. Madduri, and I. Foster. "Building scientific workflow with Taverna and BPEL: A comparative study in caGRID". In Proceedings.4th International workshop on Engineering Service-Oriented applications (WESOA), pages 118–129. 2009.

[14] D. Turi, P. Missier, C. Goble, D. De Roure, and T. Oinn. "Taverna workflows: Syntax and semantics." In International Conference on e-Science and Grid Computing, Los Alamitos, CA, USA, 2007.

[15] M. Wieczorek, R. Prodan, and T. Fahringer. "Scheduling of scientific workflows in the Askalon grid environment." SIGMOD Rec., 34(3):56–62, 2005

[16] X. Zhang, J. P. Mesirov, and D. L. Waltz. "Hybrid system for protein secondary structure prediction." Journal of Molecular Biology, 20;225(4):1049-63, 1992.