IEEE Intelligent Systems, Vol. 26, No. 1, 2011.

WINGS: Intelligent Workflow-Based Design of Computational Experiments

Yolanda Gil, Varun Ratnakar, Jihie Kim, Pedro Antonio González-Calero, Paul Groth, Joshua Moody, Ewa Deelman

Information Sciences Institute
University of Southern California

Abstract

Scientists use computational experiments to study natural phenomena through the lens of software tools and computer programs. A computational experiment consists of a description of how selected datasets are to be processed by a series of software components, in what order, and with what parameter configurations. Designing computational experiments is a challenging task for scientists because of the many interacting constraints to configure the software and to process datasets appropriately. We present a novel approach to explore the space of possible experiments efficiently starting from varying degrees of detail and varying amounts of constraints provided by the user. Our approach represents computational experiments as workflows and exploits semantic representations of datasets and software components to reason about the experiment design space. We show how the Wings workflow system can automatically track constraints and rule out invalid designs, enabling scientists to focus on the core aspects of their experiments and goals.

Computational Experiment Design

Scientists use computational experiments to study natural phenomena through the lens of software tools and computer programs [DeRoure et al 04]. These software tools can be configured with diverse settings and parameters, enabling scientists to explore different aspects of the phenomenon at hand. A *computational experiment* consists of a description of how selected datasets are to be processed by a series of software components, in what order, and with what parameter configurations. Earth scientists use computational experiments to estimate seismic hazard through simulations of earthquake forecasts. Biologists use computational experiments for analysis of gene expression microarray data or molecular interaction networks and pathways. Social scientists analyze large social networks to discover structural regularities based on mining relations among individuals.

Designing computational experiments for scientific analysis is a complex process. There are many different *software components* that could be used, ranging from general-purpose statistics packages to custom algorithm implementations to specific models and codes. Each of these software components has different requirements on the kinds of data they are designed to process and constraints on the kinds of operations they can perform. In addition, there are many accessible *datasets* that could be brought to bear in the analysis, including prototypical reference datasets as well as large shared repositories. Each dataset has *metadata attributes* that describe properties of the data that may need to be taken into account in the analysis, for example characteristics of the original collection process (eg, instrument setups) and any subsequent pre-processing and cleansing (eg, normalizing or discretizing). These metadata attributes determine what kinds of components are appropriate for processing. A given component may need to be configured according to the characteristics of the datasets to be processed. When two or more components are used in an analysis, their settings might need to be coordinated in order to obtain a valid result. Therefore, while designing a computational experiment, a scientist must choose and configure software components and datasets while considering their characteristics and requirements so that the resulting design is a valid one.

Computational experiments are becoming increasingly more complex but critical for analysis of many complicated issues. Although some may consist of a handful of steps that are previously known to the scientist, they can contain dozens of steps, be improved with novel algorithms that may be unfamiliar, and encompass aspects of data analysis that stretch beyond the field of expertise of an individual scientist. As the number of available components increases, it becomes difficult to discover both the available components and to manage the constraints that dictate how those components correctly fit together. As more and more data repositories become shareable, it is hard to keep up with available data and to track the metadata properties that affect how components are configured. To design a computational experiment, a scientist faces several challenging tasks:

1. *Design and plan an overall experiment*: Experiments can be assembled from scratch by selecting and combining components and datasets. Examples of questions that one would ask include whether datasets should be normalized before a comparison will be meaningful, or weather sampling or dimensionality reduction would be needed for a given dataset, or whether there are any labeled data available that would enable semi-supervised approaches. A common alternative to designing an experiment from scratch is to adapt experiments that others have designed. Experimenters often look up commonly utilized and widely accepted methods and reuse them as a skeleton or starting point to the experiment's design.
2. *Select specific components for the experiment:* There may be many alternative algorithms or many alternative implementations to carry out a step. For

example, to do dimensionality reduction one could use principal component analysis or linear discriminant analysis. The components may vary widely in complexity, or convenience and customization features. For example a component that implements a classifier may include also an internal feature selection method, but it may be better to use separate feature selection algorithms that may be more sophisticated.

3. *Configure components:* A given component may have parameters that can be set to different values to obtain different behaviors. Common kinds of parameters are thresholds and iteration depth. In addition, it may be difficult for a user to know how parameters should be optimally set to get better or faster results.
4. *Find datasets:* Datasets with specific characteristics may be needed for a given experiment design. For example, a semi-supervised approach may be viable only if there are appropriate labeled datasets available.
5. *Validating experiments:* For computational experiments of non-trivial nature, tracking and checking all the constraints of the software components and datasets becomes onerous. Those constraints and requirements are typically stated in technical papers and documentation, and therefore their tracking is a manual and error-prone process.
6. *Explore the space of possible experiments:* As more algorithms and datasets become available, and parameter settings, exploring the space of possible computational experiments becomes unmanageable. As a result, only a small area of the space is explored, typically focusing on already familiar components, known datasets, and default parameter values.
7. *Consider new possible experiments that incorporate new or more complex settings*: New algorithms and datasets are developed and made available constantly, and it is hard for users to keep up and incorporate them into their design. In recent years, not only new algorithms but new combinations of components have been proposed which were found to be most effective in many areas of scientific research, including ensemble methods and algorithm portfolios. These combination methods are harder to set up than single algorithms in isolation.

The approach we describe is a realization of a computer supported discovery environment [de Jong and Rip 97]. [Waltz and Buchanan 09] highlight the potential and contributions of Artificial Intelligence research over the years to computational science. They point out the increasing need for assisting scientists in designing computational experiments. While scientists will need to drive the process, many routine aspects of the design process could be automated. An intelligent system could be designed to keep track of the detailed constraints and their interactions, enabling scientists to focus on the core aspects of their experiments and goals.

Designing Computational Experiments as Workflows

In order to assist scientists with the design of computational experiments, we need a framework to represent the experiment declaratively as a composition of software components and datasets and to reason about software component constraints and relevant metadata attributes of data collections in the context of an experiment.

We use ***workflows*** to represent computational experiments. Workflows represent application components and their dependencies in terms of dataflow among them. Workflow systems have been developed to assist users with some aspect of the process, for example to assemble workflows out of large component libraries [Hull et al 06], to optimize execution performance [Wieczorek et al 05], and for workflow sharing [De Roure et al 09]. However, none of these systems provides comprehensive support for the workflow design and exploration process that we just described.

Our WINGS/Pegasus workflow system [see BOX 1] uses workflow representations to manage various aspects of workflow creation and execution. WINGS reasons about dataset and component constraints to create and validate workflows and to generate metadata for new data products [Gil et al 07; Gil et al 09]. Pegasus selects computational resources, optimizes the workflow structure, submits the workflow for execution, monitors its progress, and resolves possible failures.

BOX 1: The WINGS/Pegasus Workflow System for Scalable Distributed Scientific Analysis

Workflow systems can automate many aspects of workflow creation and execution. The WINGS/Pegasus workflow system developed at USC/ISI illustrates these capabilities, with WINGS providing automatic workflow validation and generation and Pegasus assigning resources and managing the execution. WINGS/Pegasus was initially developed to create large-scale workflows for physics-based simulations to construct seismic hazard maps for the Southern California Earthquake Center. For that application, WINGS generated a workflow with more than eight thousand computations that were mapped by Pegasus into more than twenty four thousand jobs. WINGS also generated more than 2M triples of metadata attributes for 100,000 new workflow data products.

WINGS (Workflow INstance Generation and Specialization) uses AI planning and semantic reasoners to assist users in creating workflows while validating that the workflows comply with the requirements of the software components and datasets. WINGS can reason about the constraints of the components and the characteristics of the data and propagate them through the workflow structure.

Once the workflow components and datasets are selected, Pegasus automatically maps the workflow to the available computing resources and manages its execution on local resources as well as in distributed and high-performance cyberinfrastructure resources such as the Open Science Grid (www.osg.org) and the TeraGrid (www.teragrid.org) as well as clouds. Detailed records of how new data products were generated by the system are captured in a provenance catalog, which enables easy reproducibility of results.

The WINGS/Pegasus workflow system is built on open web standards from the World Wide Web Consortium (W3C) such as the Web Ontology Language (OWL) as well as NSF National Middleware Infrastructure (NMI) software such as Condor (www.condor.org) and Globus (www.globus.org). Papers, tutorials, and open source software are available at http://wings.isi.edu and http://pegasus.isi.edu.

In this paper, we discuss novel reasoning capabilities that support the computational experiment design process. In prior publications, we have described the details of how these capabilities are implemented using semantic web technologies and grid technologies. This paper gives a user-centered perspective in terms of the assistance that users receive as part of their design process. Prior publications focused on specific details of the reasoning done by the system and how it is integrated with a robust scalable execution system based on grid computing. Thus, the key contribution of this paper is the presentation of an approach that provides assistance throughout the computational experiment design and execution process.

We have extended WINGS with novel capabilities that support the design of computational experiments as workflows. WINGS can:

- Express **reusable combinations of software components as workflow templates**
- Based on those templates, **express high-level user requests** that only partially specify what datasets, parameters, or software component types are to be used
- From a user request, **generate automatically possible workflow candidates** by searching and validating:
    - Choices of datasets
    - Choices of parameter values
    - Choices of software components
- During that search, **eliminate workflow candidates that are not viable** because they contain invalid combinations of choices

The architecture of WINGS is shown in Figure 1. WINGS operates at the domain level, where users never see any details on the execution environment. A user can design analyses by creating workflow templates. This can be done by selecting components and specifying their dataflow, much as they would do in any workflow editor. WINGS goes further in that it assists users to validate templates by enforcing the constraints specified for the workflow components. Other users could edit existing templates to create new ones. This way, the workflow template catalog would be populated with widely-used known-to-work methods that can be readily used by anyone. A user wanting to run an analysis following a known method would find the corresponding workflow template, which determines the kind of analysis to be done, and add any additional requirements particular to their analysis to create *workflow requests* or *seeds*. WINGS assists them by automatically and systematically generating possible workflows that are consistent with the request. This capability is the focus of this paper. WINGS generates workflows in three stages. First it selects components for each step in the workflow. Then datasets from the catalogs to elaborate the initial workflow request and its template. Finally it configures the parameters for each component in the workflow. Once the workflow candidates are fully elaborated, WINGS expands the workflow to specify any parallel computations over dataset collections (but not where they will take place). For all the new data products, it generates metadata attributes by propagating metadata from the input data through the descriptions and constraints specified for each of the components. The entire workflow creation and generation process is annotated in detail in a provenance catalog for later inspection. WINGS can convert any valid and complete workflow into a format for submission to an execution engine. Currently WINGS can submit workflows in a scripted format for execution in the local host, or submit workflows to Pegasus which will manage their execution in shared distributed resources in an efficient and scalable manner.

WINGS uses W3C's OWL, RDF, and SWRL (http://www.w3.org/2001/sw) to represent workflows and their associated constraints. The core classes are defined in OWL, and workflow templates, requests, and candidates are expressed in RDF. Many constraints are represented as RDF triples, others that are more complex are represented in SWRL.

WINGS assumes external catalogs of software components and datasets that can be accessed through services. This is an important feature, since scientific environments consist of distributed services to access the data and algorithms necessary for data analysis. Many collaborations are set up to enable each institution to contribute resources (data, instruments, computers, software) by making them available to others while remaining under the control of the home institution. Examples include the National Virtual Observatory (http://www.us-vo.org), the Earth System Grid (http://www.earthsystemgrid.org), the Biomedical Informatics Research Network (http://www.nbirn.org), and the Cancer Biomedical Informatics Grid (http://cabig.cancer.gov). Data providers expose services to access data sources. There can be many organizations playing the role of data providers, and as a result data may be accessible in various catalogs that are distributed across the network. Other organizations may provide algorithms, services, models, or implemented codes that can process data and can be used as components of workflows. Each provider maintains the resources they contribute. WINGS assumes that the workflow system can query these catalogs to reason about their contents. This is an important feature, and we will come back to discuss it on a later section.

To illustrate our approach we use very simple examples from machine learning and data mining, since machine learning algorithms are widely used in computer science and other sciences and are likely to be familiar to the reader. It is also a domain that illustrates the distributed nature of data and algorithm providers. Many algorithms for machine learning are available in libraries such as Weka (http://www.cs.waikato.ac.nz/ml/weka) and MLC (http://www.sgi.com/tech/mlc), while many machine learning datasets used by researchers are maintained in the well-known Irvine repository (http://archive.ics.uci.edu/ml). Many other research groups make datasets and algorithms available on their own project sites. As in many other scientific disciplines, researchers seek and marshal data and algorithms for their computational experiments from these disparate and distributed resources.

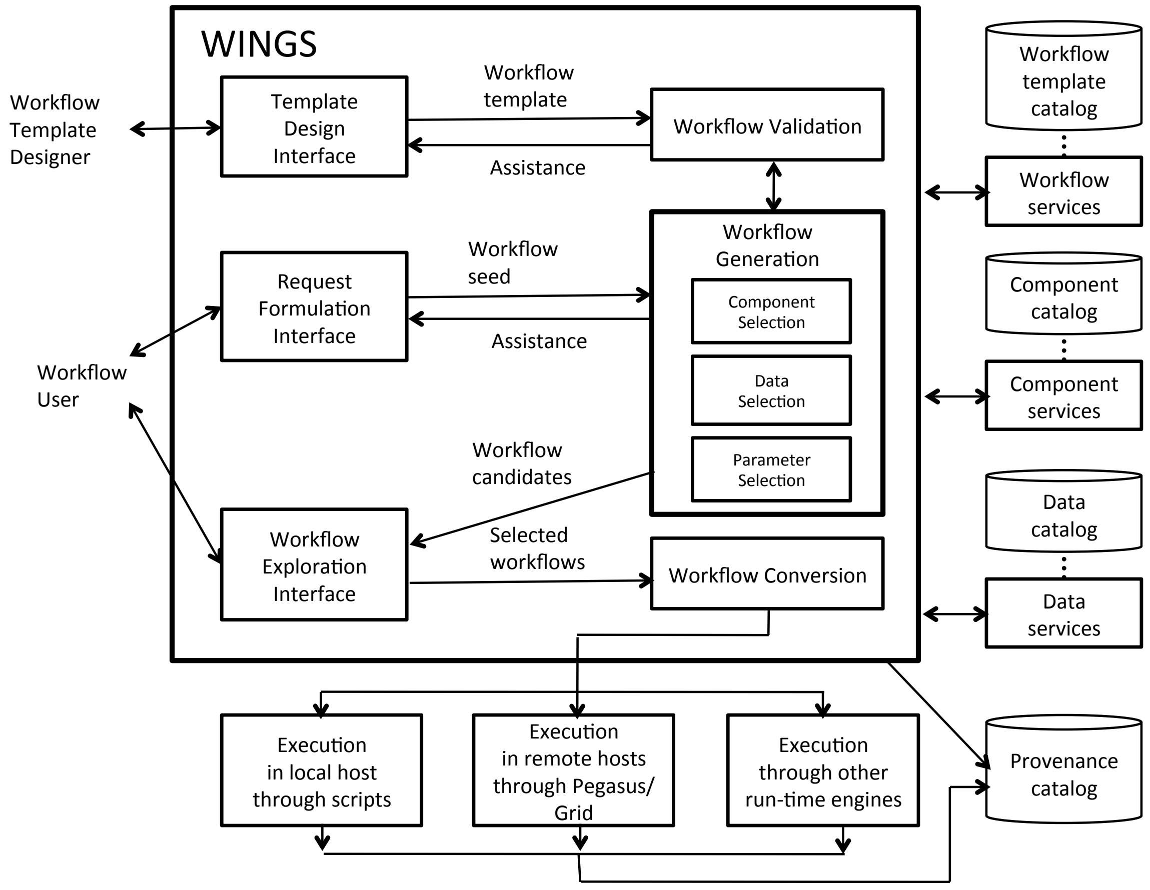
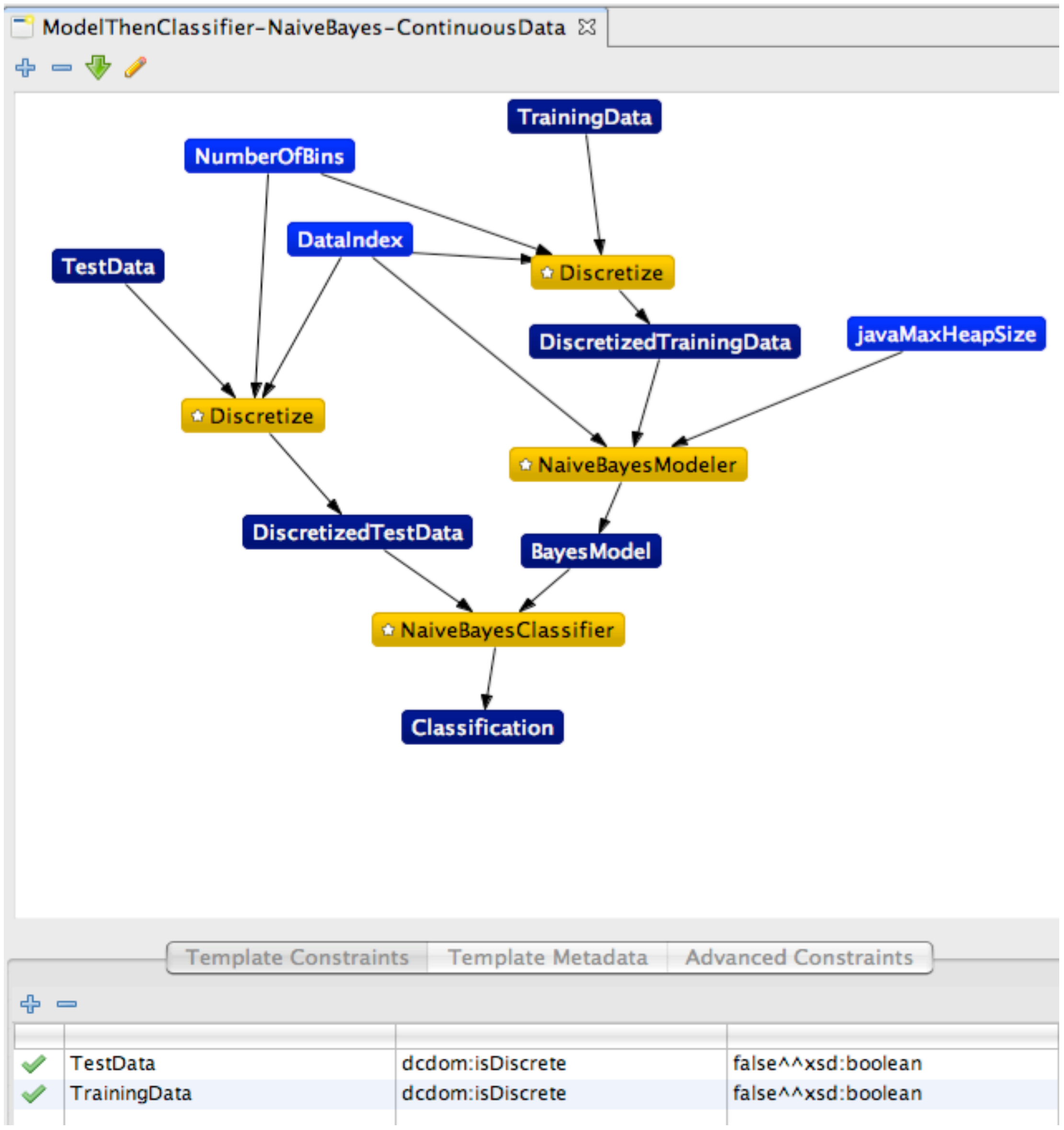


Figure 1: Architecture of the WINGS workflow system.

Designing Reusable Methods

Workflow templates are high-level reusable analysis methods for computational experiments. Only some or none of the input datasets may be specified, and only some or none of the component parameters may be set.

WINGS represents a workflow as a set of variables. Each component, dataset, and parameter occurrence is assigned a variable. A workflow in WINGS is a complex RDF structure with assertions about those variables and the overall dataflow. The WINGS interface shows two views of the workflow structure: a *dataflow graph* that shows the connections across components, and a *constraints table* that shows triples of attribute values applied to workflow variables. The variables are shown in the dataflow graph, and have a marking when they are assigned a specific value. The constraints table refers to those variables to specify values, types, and other constraints.



✓	TestData	dcdom:isDiscrete	false^^xsd:boolean
✓	TrainingData	dcdom:isDiscrete	false^^xsd:boolean

Figure 2: Example of a workflow template in WINGS.

Figure 2 shows a workflow template in WINGS to classify a continuous dataset with a Naïve Bayes algorithm. The constraints tab at the bottom shows that the workflow requires that input datasets be continuous. In some templates, the components are classes rather than specific components. We mark variables that have a specific value with a small white star. For example, Discretize and NaïveBayesModeler are specific components and are marked with a star, while a template with a step Modeler refers to a class of components and would not be marked with a star (we will see an example in the next section). Some datasets can be pre-selected, and some parameters can be pre-set.

Workflow templates may also include advanced constraints that are expressed as rules, and shown in the Advanced Constraints tab of the interface. For example, a well-known principle in ML is that test data cannot be used for training. In WINGS, a workflow that trains a classifier and runs the model on test data can be constrained so that the test data are never included in the training dataset. This is done with the following rule:

```
[modelerInput_not_equal_to_classifierInput:
     (:modelerInput wflow:hasDataBinding ?ds1)
     (:classifierInput wflow:hasDataBinding ?ds2)
     equal(?ds1, ?ds2)
     (?t rdf:type wflow:WorkflowTemplate)
          > (?t wflow:isInvalid "true"^^xsd:boolean)]
```

Some constraints are expressed de facto in the template diagram. In our example, the parameter number of bins used in the two discretizers is constrained to have the same value. This is a result of the two components sharing the same parameter variable as input.

Templates can be created from existing templates. For example, the workflow template on the right side of the figure was created from a generic sample-then-model-then-classify workflow template by adding a constraint to specify the value of the sampling interval parameter. To add constraints, a user adds a workflow variable to the table, and then WINGS shows a list of properties that apply to that variable. Once the property is selected, the user can specify a type constraint or provide a value. Users can also add components to the workflow and specify their dataflow links. Users can also create workflow templates from scratch, by selecting components and linking them and specifying constraints on the table. Each template keeps associated metadata about its provenance, which is shown in the Metadata tab.

WINGS validates a workflow template by merging any constraints expressed by the designer with constraints about datasets and components that are retrieved by the workflow system from the respective catalogs. This is done using the workflow generation and validation algorithm described below. Through this validation mechanism, WINGS ensures that workflows adhere to correct practices by using algorithms and datasets properly.

Three important unique properties of how WINGS represents workflows are key to assisting users in the design of computational experiments:

1. A workflow template is an abstraction of many possible computational experiments consistent with the given design as represented by the template. The ability to use abstract components in a workflow allows users to sketch out their initial thoughts about the experiment without being forced to decide on specific algorithms up front. In creating a workflow template, users do not have to specify what data or parameters will be used.
2. The designer of an experiment can communicate its applicability and intended use as constraints in the workflow template. When reusing or adapting a template, the new template can be validated because the

original one already specified constraints that can be checked in the new design.

3. Users less experienced in designing computational experiments can build on the expertise of others by reusing their workflow templates and modifying them to create new workflows, which are automatically validated by the system to ensure appropriate use of the components.

In summary, users are able to create or find high-level reusable designs that match their ideas about how a computational experiment should be carried out while getting assistance from the system in creating valid designs that are consistent with the system's prior knowledge about other workflows and specifications of the software components.

Specifying Experiments

Workflow seeds are user requests for creating an executable workflow. A seed is formed by a workflow template combined with additional type constraints, parameter configurations, or dataset selections. In WINGS, a seed is an underspecified request that the system automatically elaborates and completes for the user.

Figure 3 shows an example of a seed. It uses a template that discretizes datasets first and then runs a modeler and a classifier. To this template, the user added a constraint that the testing dataset needed to be labor-2007-07-30-195521, which contains labor data statistics in a specific region. The user also constrained the model to be human readable (decision tree models are but Bayes models are not). Both constraints are shown at the top of the screen as a constraints table for the seed. Like templates, seeds can have Advanced Constraints and Provenance Metadata. For example, the user specified a constraint that the training and test datasets must be from the same region.

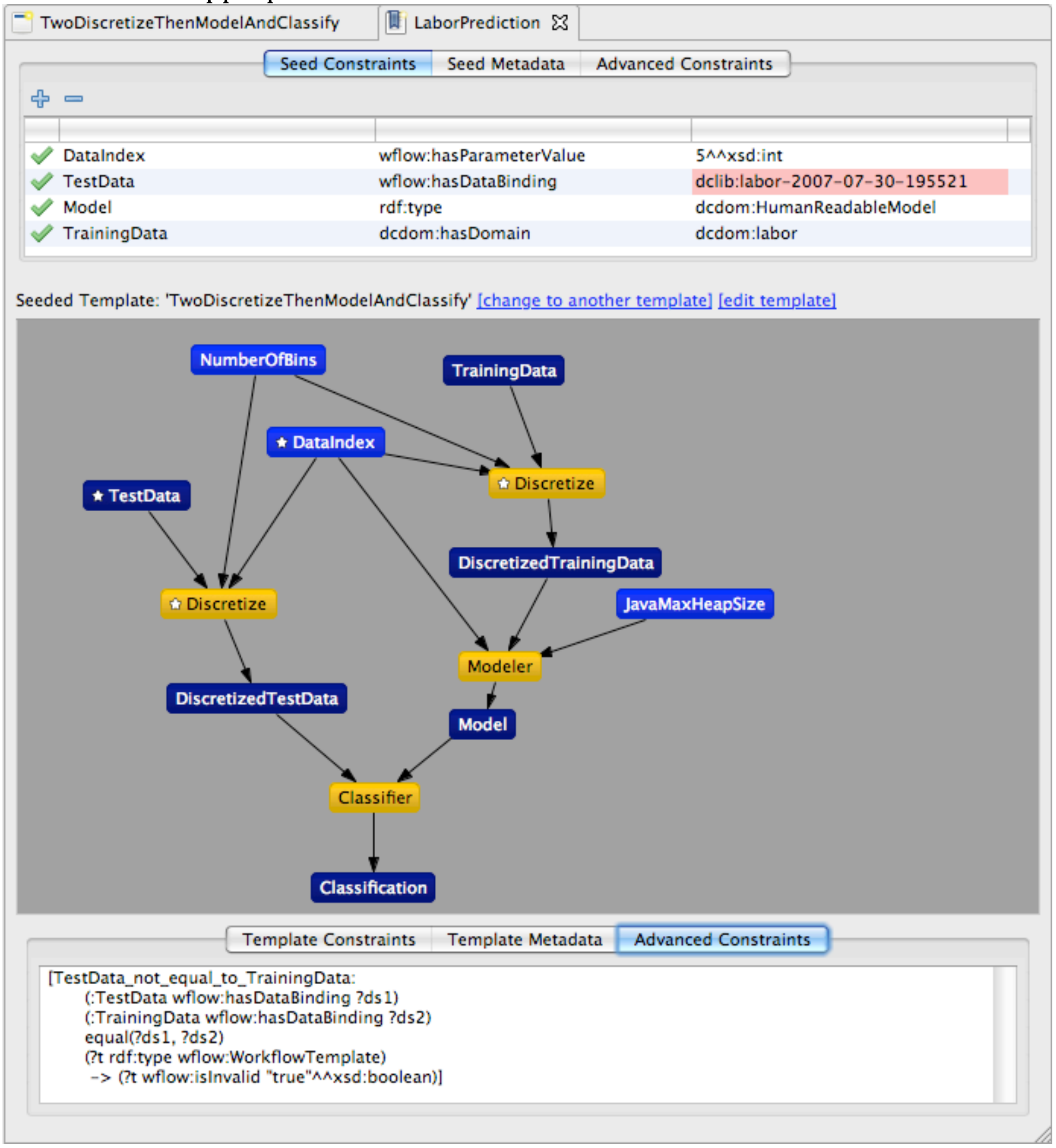
Rather than designing analyses from scratch, users can design analyses by reusing workflow templates developed by other users. In this example, the user reused an existing generic modeler then classifier workflow template whose input data must be discrete. Users can also modify a workflow template from the library to create a customized analysis, as well as contribute their own templates to the library.

Note that while a template and a seed are conceptually different, they use the same underlying representation. The only difference is how they are categorized by a user either as a general reusable template or as the specification of an experiment respectively.

The distinction between seed workflows and template workflows is important. Users do not have to design analyses from scratch, rather they can use the available high-level data analysis routines (captured as templates) as the start of their analyses. Seeds may leave the following unspecified:

- Algorithms to be used. In our example seed, "modeler" and "classifier" are classes of algorithms. The user is essentially requesting that the system finds appropriate algorithms for those workflow steps.

- Datasets to be used. In our example seed, the user did not specify an input training dataset for "TrainingData". The user is requesting that the system finds an appropriate dataset.



✓ DataIndex	wflow:hasParameterValue	5^^xsd:int
✓ TestData	wflow:hasDataBinding	dclib:labor-2007-07-30-195521
✓ Model	rdf:type	dcdom:HumanReadableModel
✓ TrainingData	dcdom:hasDomain	dcdom:labor

Seeded Template: 'TwoDiscretizeThenModelAndClassify' [change to another template] [edit template]

```
[TestData_not_equal_to_TrainingData:
     (:TestData wflow:hasDataBinding ?ds1)
     (:TrainingData wflow:hasDataBinding ?ds2)
     equal(?ds1, ?ds2)
     (?t rdf:type wflow:WorkflowTemplate)
  -> (?t wflow:isInvalid "true"^^xsd:boolean)]
```

Figure 3: A workflow seed requesting classifications for a labor dataset with a constraint that the model be human readable.

- Parameter settings. In our example, the javaMaxHeapSize is not specified and the system is expected to set its value.

Because seeds can be underspecified, users are not forced to detail every last algorithm, parameter, or dataset. Instead, users can focus on the requirements that are most pertinent to their problem.

When a workflow seed is underspecified in any way, WINGS elaborates the seed into workflows that can be submitted to an execution engine. The next section describes the algorithm for generating workflows and walks through an example of how this happens. Workflow seeds can be fully specified and if so WINGS simply checks the validity of the seed in case the user provided any incompatible constraints and transforms the workflow into a format suitable for its underlying execution engine.

Automatic Generation of Valid Experiments

WINGS generates valid workflow candidates that are consistent with a given workflow request. The workflow generation algorithm has three major phases, summarized in BOX 2. The algorithm queries the data and component catalogs and retrieves constraints that are merged and propagated throughout the workflow structure, first from results to inputs and then from inputs to results. In all three phases, any workflow candidate whose assignments are inconsistent is considered invalid and therefore eliminated from the candidate set. Only consistent choices of datasets, components, and parameter values lead to valid candidates, which WINGS can then translate into a format suitable for a workflow execution engine. The algorithm is also used to validate any workflow, including templates and requests, by detecting whether it leads to inconsistent constraints in any of the phases.

BOX 2: AUTOMATIC WORKFLOW GENERATION AND VALIDATION IN WINGS

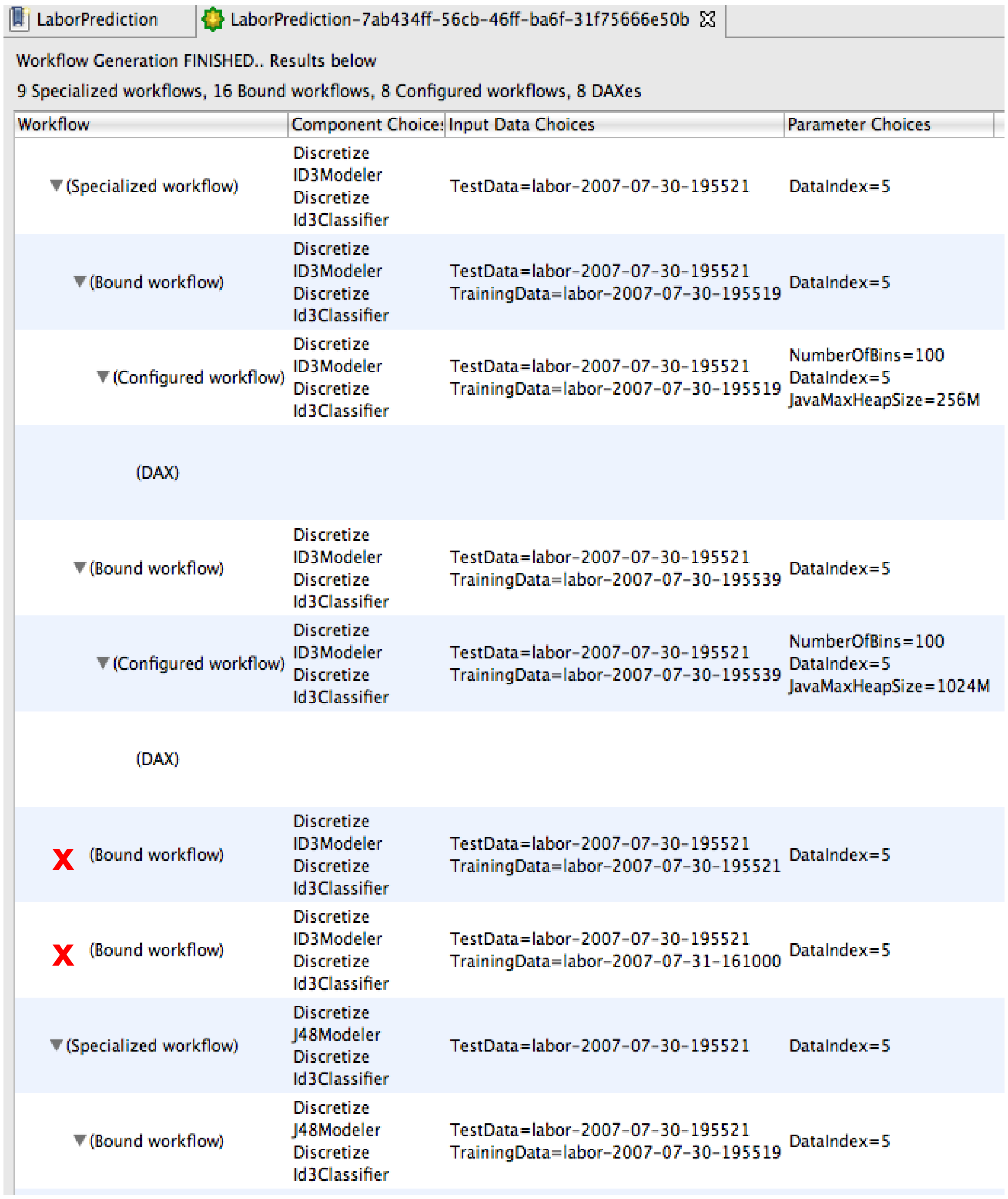
From an underspecified user request, WINGS generates workflow candidates that are consistent with the request and completes their specification so they can be submitted for execution. In the process, WINGS validates each workflow candidate and eliminates those that contain inconsistent constraints for datasets or components. The algorithm elaborates workflow candidates in three major phases:

Phase 1. Component selection: Starting from the end data products in the workflow, the workflow is traversed backwards and for each component find additional constraints on any input arguments given the constraints on output. If an abstract component is encountered in the workflow, then the component catalog is asked for a set of specialized components that match the current workflow constraints. A new workflow candidate (a *specialized* workflow) is created with each specialized component, and the backward traversal proceeds. Some parameter values may be set during this step as more constraints on arguments are introduced. At the end of this step, the constraints have been propagated to the workflow inputs for each workflow candidate.

Phase 2. Dataset selection: Given the constraints on inputs from the prior step, the data catalog is queried to find available datasets that match those constraints. There can be more than one dataset combinations for a given input set. For each combination of datasets, a new workflow candidate (a *bound* workflow) is created by binding the input data variables to matching data objects. Candidates with no matching data sources will be rejected as invalid. The data catalog is then queried for additional metadata properties for each dataset in a workflow candidate, which will be used in the next step.

Phase 3. Parameter selection: Given the metadata properties of the input datasets from phase 2 and their additional constraints propagated in phase 1, traverse the workflow forward from the initial inputs and for each step query the component catalog for additional constraints on its outputs given the constraints on its inputs. This step results in workflows that contain metadata properties for all intermediate and final workflow data products. For each step, the component catalog is also queried regarding parameter values that are appropriate given the constraints on input and output datasets. For each choice of parameter settings, a new workflow candidate (a *configured* workflow) is created. Candidates are rejected if no parameter settings are possible, or if the component constraints are incompatible with its input data constraints.

Only valid and completely specified workflows are generated with this algorithm. The user can select



LaborPrediction | LaborPrediction-7ab434ff-56cb-46ff-ba6f-31f75666e50b

Workflow Generation FINISHED.. Results below

9 Specialized workflows, 16 Bound workflows, 8 Configured workflows, 8 DAXes

Workflow	Component Choices	Input Data Choices	Parameter Choices
▼ (Specialized workflow)	Discretize ID3Modeler Discretize Id3Classifier	TestData=labor-2007-07-30-195521	DataIndex=5

▼ (Bound workflow)	Discretize
	ID3Modeler
	Discretize
	Id3Classifier
		TestData=labor-2007-07-30-195521
		TrainingData=labor-2007-07-30-195519
			DataIndex=5

▼ (Configured workflow)	Discretize
	ID3Modeler
	Discretize
	Id3Classifier
		TestData=labor-2007-07-30-195521
		TrainingData=labor-2007-07-30-195519
			NumberOfBins=100
			DataIndex=5
			JavaMaxHeapSize=256M

(DAX)			

▼ (Bound workflow)	Discretize
	ID3Modeler
	Discretize
	Id3Classifier
		TestData=labor-2007-07-30-195521
		TrainingData=labor-2007-07-30-195539
			DataIndex=5

▼ (Configured workflow)	Discretize
	ID3Modeler
	Discretize
	Id3Classifier
		TestData=labor-2007-07-30-195521
		TrainingData=labor-2007-07-30-195539
			NumberOfBins=100
			DataIndex=5
			JavaMaxHeapSize=1024M

(DAX)			

✗ (Bound workflow)	Discretize
	ID3Modeler
	Discretize
	Id3Classifier
		TestData=labor-2007-07-30-195521
		TrainingData=labor-2007-07-30-195521
			DataIndex=5

✗ (Bound workflow)	Discretize
	ID3Modeler
	Discretize
	Id3Classifier
		TestData=labor-2007-07-30-195521
		TrainingData=labor-2007-07-31-161000
			DataIndex=5

▼ (Specialized workflow)	Discretize J48Modeler Discretize Id3Classifier	TestData=labor-2007-07-30-195521	DataIndex=5

▼ (Bound workflow)	Discretize
	J48Modeler
	Discretize
	Id3Classifier
		TestData=labor-2007-07-30-195521
		TrainingData=labor-2007-07-30-195519
			DataIndex=5

a subset and WINGS converts them to a format appropriate for the execution engine of choice.

Figure 4: Workflow candidates created during workflow generation.

Figure 4 shows some of the workflows generated by WINGS for the request in Figure 3. A total of 8 workflow candidates are found to be valid for the given request. In the figure, they are shown as configured workflows, which have all datasets selected and all parameter values set. Each has an associated DAX for submission to Pegasus for execution. We now explain phase by phase how the algorithm generates these candidates.

Phase 1: Component Selection

In the first phase of the WINGS generation algorithm, components are selected. If the workflow request contains abstract component classes, then the component catalog is queried about valid specialized components for that class. An alternative workflow candidate is created for each possible specialization.

In our example, WINGS specialized the workflow template by finding 9 possible combinations of modelers and classifiers that would work with the constraints of the workflow request. In this case, no Bayes modelers or classifiers were used since they respectively generate or use a Bayes model and the request specified to use a decision tree model. For our example, we had 3 decision tree modelers and 3 decision tree classifiers available (for ID3, J48, and Lmt, all well-known decision tree algorithms). Figure 4 shows 2 of the 9 possible combinations of modelers and classifiers, namely an ID3 modeler with an ID3 classifier and an ID3 modeler with a a J48 classifier. We will see in later phases that some of these 9 combinations are not viable for the request and will get eliminated.

Also in this phase, the component catalog is queried about additional constraints placed on a component's inputs given the constraints on its outputs. Constraints are propagated as the algorithm does a backwards traversal from results to input data throughout the workflow. This allows WINGS to have an extended set of constraints for input datasets that are used in the next phase for dataset selection.

In our example, a query about each modeler returns a constraint that its input model must be on the same domain as the test dataset. In this case, the domain of the model must be based on labor data (specified as having labor as its domain). A query about each classifier returns a constraint that its input training dataset must be of the same domain as the model it outputs. Therefore, the training dataset that is input to the workflow includes a constraint that it must be labor data (has labor as its domain).

Phase 2: Dataset Selection

In the second phase, datasets are selected by querying the component catalog about datasets that match the constraints in the workflow request for the input datasets found in phase 1. An alternative workflow candidate is created for each possible dataset match.

In our running example, only the training dataset is unspecified and the test dataset is given. As we saw from the prior phase, WINGS has now found that the input training dataset has a constraint that it must be of the labor domain. The data catalog we had for this example has 4 datasets for labor data, and all of them have some missing values (i.e. hasMissingValues is set to true). All workflows which have Lmt algorithms (classifiers or modelers) require no missing values, so we only get the 4 dataset bindings for the workflows without Lmt algorithms. In the figure, two candidate bound workflows are eliminated and shown with a red mark. Thus we end up with a total of 16 candidates that have bound inputs (4 for each of the 4 algorithm combinations that do not have Lmt algorithms).

In this phase, the data catalog is also queried to return all metadata properties of the datasets in each workflow candidate. This is also done for any datasets specified in the request, so that all the properties of those datasets are taken into account as constraints are propagated in the next phase of the algorithm. In our example, a metadata property would be returned for each labor dataset stating its size.

Phase 3: Parameter Selection

In the third and final phase, the constraints from the input datasets are propagated forward in the workflow through each component by querying the component catalog regarding metadata constraints on its outputs given the constraints on its inputs. Any workflow candidates that have inconsistent constraints are eliminated as invalid. Also in this phase, parameter values are selected by querying the component catalog regarding appropriate parameter values for a component given the metadata properties of its input and output datasets. Alternative workflow candidates are created for each of the possible parameter assignments.

In our example, the component catalog is queried for a value for the number of bins for the first discretizer and is given the size of the training datasets. The value returned in this case is 100. The component catalog is given this value for the second discretizer and it is found to be acceptable. The parameter for the java heap size (exposed by the software in the Weka library) is also set for each modeler based on the size of its input datasets.

During this phase, a total of 8 of the 16 workflow candidates are eliminated. In propagating the constraints through the workflow in a forward traversal, some dataset choices from phase 2 are found to be inappropriate. First, the workflow template had a constraint that the test data set and the training dataset could not be the same, so 4 workflow candidates are eliminated. Second, the request had a constraint that the training and test datasets must be from the same region and as a result 4 more workflow candidates are eliminated. In this example no workflow candidates for parameter value choices are eliminated, but this could happen for example if the classifier found the value for the number of bins unacceptable for the test dataset. In the end, a total of 8 workflows are found to be viable for the workflow request in our example.

Discovery and Exploration

While we have used machine learning workflows in our examples, WINGS has been used in several scientific applications including earthquake science, biomedical image processing, and genomics [Gil et al 07; Kumar et al 09]. In particular, in biomedical imaging processing choice of datasets and parameters in the computational experiment affect the quality of the results as well as the performance of the system. By breaking the image into chunks that can be processed in parallel, better performance is obtained but the results are less accurate. In those applications, WINGS is used to explore these tradeoffs in the design of the workflows. The workflows generated by Wings vary as the system explores different choices of datasets and parameters. More details can be found in [Kumar et al 09].

A very important result of the workflow generation process that the user may discover a combination of components that they were not previously familiar with, or find data sets that they did not know were compatible with a particular experimental design. Furthermore, WINGS will always include new algorithms and datasets as they become available. For example, it could be set to periodically check updates relevant to selected computational experiments and alert the user if new relevant datasets or components are found.

The result of automated workflow generation is not only a set of valid workflows that can be submitted for execution, but also a range of possible experiments with different levels of specialization. For example, there are candidates that only contain concrete components but not specific datasets. Likewise, there are candidates that include both concrete components and specific datasets but do not have parameter values assigned. These candidates may be useful as starting points for other experiment explorations. WINGS allows any of the candidates to be saved as templates or seeds for later reuse and modification.

For situations when many candidate workflows are generated for the user, we envision a series of interactive iterations over the workflow generation process where the user could browse the candidates and formulate further constraints in their requests until a reasonable set of candidates is obtained.

Conclusions

This paper presented our approach to assist users in designing and exploring computational experiments as workflows. The underlying workflow reasoning algorithms reason about constraints on data and components and propagate those constraints throughout the workflow structure. Throughout the interaction the workflow developer is only shown the effect that those constraints have in the context of their particular workflow. Moreover, the workflow developer need not keep track of such constraints because they are automatically incorporated into their workflow by the system and propagated during workflow generation and validation. This makes it possible to *modularize* scientific knowledge: the knowledge that a workflow designer needs to use is separate from the knowledge that an algorithm designer has. Once an algorithm designer contributes an algorithm to the catalog and expresses his/her knowledge about the component requirements and constraints, this knowledge is readily exploited by the workflow system and used to assist all designers of computational experiments that could benefit from that algorithm.

A promising area of future work is integrating workflow execution into the workflow design process. Some workflows need to be designed in portions, where looking at the actual results of a portion helps decide how that portion should be and once it is settled then the next portion can be designed. Another interesting area of future work is the design of appropriate mechanisms to expose the system's reasoning to a user. For example, a user may want a justification for why a parameter was set to a certain value or the reason why an algorithm is not appropriate for their data. This would involve both extending the system to record the reasoning behind its decisions as well as designing the presentation of such information.

Scientists often speak of a "data deluge", and are well aware of the need and benefits of shared data catalogs that expose important metadata relevant to discern appropriate use of the data and to facilitate data integration. There is an analogous deluge of algorithm and analytic tools, and no practical mechanisms to facilitate their use and integration. Scientists need intelligent assistance in finding algorithms that are appropriate for their data, in discovering recent algorithms that they are not familiar with but are relevant for their experiment, and in configuring a valid workflow that complies with all the interacting constraints among datasets and analytic tools. Workflow systems equipped with semantic reasoning capabilities can enable exciting prospects for assisted scientific discovery.

Acknowledgements

We gratefully acknowledge funding from the National Science Foundation under grant CCF-0725332 and by the Spanish Ministry of Science under grant TIN2006-15202-C03-03. We thank our many scientist collaborators who motivated the research described in this paper.

References

de Jong, H. and A. Rip. The computer revolution in science: steps towards the realization of computer-supported discovery environments, Artificial Intelligence, Special Issue on Scientific Discovery, Volume 91, Issue 2, Pages 225-256, 1997.

DeRoure, D., Gil, Y., and Hendler, J. Guest Editor's Introduction to the Special Issue on e-Science, IEEE Intelligent Systems, January 2004.

De Roure, D., Goble, C. and Stevens, R. (2009) The Design and Realisation of the myExperiment Virtual Research Environment for Social Sharing of Workflows. Future Generation Computer Systems 25, pp. 561-567.

Gil, Y., Ratnakar, V., Deelman, E., Mehta, G. and J. Kim. "Wings for Pegasus: Creating Large-Scale Scientific Applications Using Semantic Representations of Computational Workflows." Proceedings of the 19th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI), Vancouver, British Columbia, Canada, July 22-26, 2007.

Gil, Y., Groth, P., Ratnakar, V. and C. Fritz. "Expressive Reusable Workflow Templates." Proceedings of the Fifth IEEE International Conference on e-Science, Oxford, UK, December 9-11, 2009.

Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M., Li, P., and T. Oinn. "Taverna: A Tool for Building and Running Workflows of Services", Nucleic Acids Research, Vol 34, 2006.

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

Waltz, D. and Buchanan, B. G. "Automating Science." Science, Vol. 324, April 2009.

Wieczorek, M., R. Prodan, and T. Fahringer, "Scheduling of Scientific Workflows in the ASKALON Grid Environment," SIGMOD Record, vol. 34, 2005.