Proceedings of the Seventh IEEE International Conference on Semantic Computing (ICSC), Irvine, CA, 2013.

Mapping Semantic Workflows to Alternative Workflow Execution Engines

Yolanda Gil
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
+1-310-822-1511
gil@isi.edu

Abstract—**Current workflows have limited reusability and shelf life because they specify particular application software to be run at each step, which can become obsolete or no longer run. We describe semantic workflow representations composed of domain tasks, which the system then automatically maps to the application codes available in the particular execution environment. We have implemented this approach in the WINGS semantic workflow system, and demonstrate the mapping of workflows to different sets of application codes depending on the workflow execution engine and execution environment selected.**

***Keywords-** Scientific workflows; semantic workflows; abstract workflows; workflow reuse*

I. Introduction

A variety of workflow systems have been developed to manage complex scientific computations [11]. An important area of research has been to allow users to specify workflow computations in a manner that is independent from the execution environment, where the workflow system automatically maps the codes to whatever execution resources are available at run time. This can be seen as managing a separation between the physical layer and the logical layer of the computation. A workflow specification at the logical layer has a specification of the codes to be executed, and there is no mention of the actual resources where the execution will take place. A workflow specification at the physical layer does mention execution resources that are to be used to run the computations. This separation between the logical and the physical layers is also common in web services frameworks and has been adopted in some workflow systems [10]. A workflow specification at the logical layer is sometimes called an "abstract workflow" [2]. Workflow systems automatically map the workflow specification at the logical layer to a workflow specification at the physical layer. This is an important benefit, as it enables users to run their workflow in different execution environments, bringing flexibility to their applications.

However, workflow representations are still very tied to the execution environment because they specify the application codes that are to be run at each step. For example, depending on the workflow system a step may specify the MATLAB routine or Java code to run, or the signature of the service that needs to be invoked. In this respect, workflows are still tied to particular application codes and software environments. When published in workflow repositories [1], this limits their reuse by others who may use different software. It also limits their validity over time when code becomes obsolete and no longer runs. Ideally, workflow specifications would be independent of the particular code and software environment, specifying only the domain task to be carried out rather than what application codes to run. Previous work has focused on interoperability of workflow systems and workflow representations [8], but not on creating more abstract representations that address the domain layer.

We have developed an approach to represent workflows that express domain tasks rather than the application codes that implement those tasks. This paper describes how we tackle three important challenges: 1) representing domain tasks, 2) matching invocation signatures of application codes to domain tasks, 3) mapping workflows of domain tasks to workflows of application codes that can be executed in alternative workflow execution engines. We have implemented this approach in the Wings semantic workflow system, and extended Wings to demonstrate the mapping of semantic workflows into two workflow execution engines: Pegasus/Condor [2] and Apache OODT [9].

The paper begins by introducing domain tasks, and the benefits of semantic workflows that represent domain tasks. We then discuss our approach to represent domain tasks, and their relationship to application codes. We describe our approach to automatically map semantic workflows composed of user-defined domain tasks into application codes available in different execution engines.

II. Workflows at the Domain Layer

Data analysis tasks are implemented by application codes. They represent abstractions of the functions that those codes perform. An example of a domain task is linear regression, which could be implemented with a variety of application codes such as a MATLAB routine, an R routine, a Java code, etc. A semantic workflow specification at the domain layer would have steps that specify domain tasks. A semantic workflow system would then map automatically the workflow specification at the domain layer into a workflow specification at the logical layer. A workflow execution engine would, in turn, map the logical layer (application codes) to the physical layer (i.e., execution resources).

Semantic workflows at the domain layer are closer to specifications of scientific methods that are described in scientific publications. These methods describe steps in ways that are not dependent on the application codes, and are therefore more reusable by other researchers who may use different software.

Workflow specifications at the domain layer would provide scientists with additional flexibility:

- **Facilitate method design and testing**: Scientists would design semantic workflows at the domain layer, and submit the same workflow in a local host or in a shared resource. For example, scientists often use smaller datasets to test different workflow designs on a local host, then when one of the workflow designs looks promising it can be submitted to execution with larger datasets in shared high end computing resources. Their local resource and the shared resource may not have the same codes. For example, a shared resource may have an efficient MPI version of a clustering algorithm, while a local host may have an inefficient one that is easy to install locally and is sufficient for testing purposes.
- **Provide new failure recovery mechanisms**: When the execution of a particular application code fails, the semantic workflow system could select an alternative application code for the same domain task specified in the step that failed.
- **Portability and reproducibility of methods across research groups**: Different research groups often use different software environments for their analyses. Some research groups prefer to use proprietary software for their work (e.g., MATLAB), perhaps because it is more reliable or more efficient. Other groups prefer using open software that implements similar functionality. Other labs prefer to develop their own software. A semantic workflow specified at the domain level could be more easily transferred and reused across research groups.
- **Archival publication of methods**: The software base that is used today may not be available tomorrow. Software libraries evolve, implementations are revised into new versions, and commercial software has new periodic releases. Libraries and packages are abandoned in favor of new ones. Describing a semantic workflow at the domain layer enables the method to survive the test of time more easily, making workflows more portable over time.
- **Improve provenance and metadata annotations**: Workflow systems automatically annotate provenance and metadata of workflow data products. If the provenance annotated is specific to the application codes then it has limited generality. A semantic workflow system would record provenance in terms of domain tasks independent of the codes executed.
- **Facilitate the transition of methods from research into operational environments**: A research environment may use more exploratory application codes, while an operational environment may use more robust and efficient implementations. Transitioning methods from research to operations could be facilitated with semantic workflow specifications at the domain level.

III. Semantic Worfklows of Domain Tasks

Our work addresses three major challenges in providing semantic workflow specifications at the domain layer:

1. Representing domain tasks with sufficient generality to encompass equivalent application codes and with sufficient detail to provide meaningful descriptions to scientists.
2. Mapping domain tasks into application codes.
3. Automating the generation of executable workflows from workflows of domain tasks.

A. *Representing Domain Tasks*

We create representations of domain tasks using ontologies. We define a concept (class) for each domain task. Each of the arguments of a task is represented as a property of that class. When used in a workflow, the dataset or parameter for an argument is the value of the corresponding property. These classes form ontologies of domain tasks, containing hierarchies that organize them from more general abstract ones to more specific ones. These ontologies form a *Catalog of Domain Tasks (CDoT)*. The domain tasks in the CDoT are then used to represent workflow steps at the domain layer.

Figure 1 illustrates the representation of domain tasks in a CDoT for machine learning, described in [7]. Domain tasks for machine learning such as modeling and classification, shown in the top part, can be done using different approaches, such as decision trees and Bayesian methods. Two decision tree algorithms are shown: ID3 and J48, each represented as a subclass. Training a classifier using ID3 is a domain task, represented in this CDoT as "ID3-Modeler". This domain task can be implemented in codes in different languages and libraries, and all can be used to execute that task as we describe next.

B. *Mapping Domain Tasks to Application Codes*

Domain tasks are ultimately implemented by application codes. We assume that a workflow execution engine has a *Catalog of Application Codes (CAC)*, concerned with the logical layer and that includes implemented codes available to run for individual workflow steps. Therefore, we need a mechanism to map domain tasks in semantic workflows into application codes.

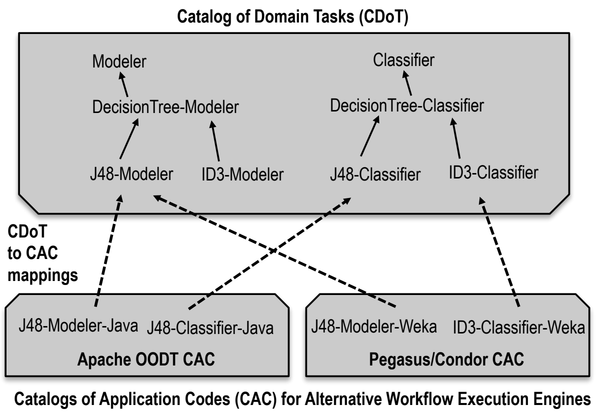


Figure 1. The Catalog of Application Codes (CAC) for each workflow execution engine is mapped to the Catalog of Domain Tasks (CDoT).

Our approach to facilitate the mapping of semantic workflows to application codes available in an execution engine is through the use of ontologies. Application codes in the CAC are represented as a class similar to the representation of domain tasks. A CAC must always refer to a CDoT, which can be done through importing the CDoT ontology. Then, each application code available in a workflow execution engine's CAC is mapped to a domain task concept in the CDoT through a subclass relation.

Figure 1 illustrates this, showing the CACs for two different workflow execution engines. There are different application codes for the J48 and ID3 modelers and classifiers available in each workflow execution environment.

The invocation signatures of domain tasks need to be mapped to the signatures of application codes. In our work to date we constrain this problem by making a simplifying assumption. We assume that each domain task has the same signature as the application codes that implement it. We prepare and encapsulate application codes so that their parameters are always aligned with the domain task that they accomplish.

C. Generating Executable Workflows from Workflows of Domain Tasks

In prior work, we developed workflow reasoning algorithms that automatically specialize high-level workflows so that each step is mapped to executable components to create a workflow specification at a logical layer [6]. We extended this work so that there is a clear separation between the CDoT and the CAC for the algorithm. The user creates workflows using domain tasks from the CDoT. To run a workflow, the user can then select one of the CACs available for that CDoT. Each CAC is supported by a workflow execution engine that can run those application codes in its execution environment. The workflow reasoning algorithms then generate executable workflows by mapping domain tasks in the CDoT to the application codes available in the selected CAC.

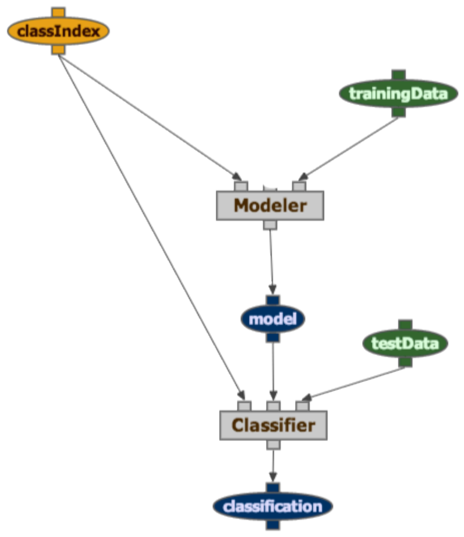
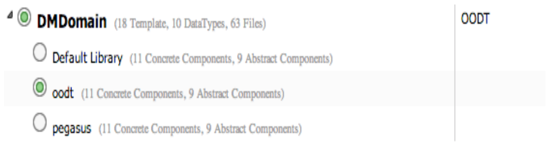


Figure 2. A semantic workflow is defined using component classes from the CDoT, which is independent of the workflow execution engine as well as the application codes available within its execution environment.



DMDomain (18 Template, 10 DataTypes, 63 Files) | OODT
Default Library (11 Concrete Components, 9 Abstract Components)
[✓ oodt] (11 Concrete Components, 9 Abstract Components)
pegasus (11 Concrete Components, 9 Abstract Components)

Figure 3. Selecting the OODT workflow execution framework.

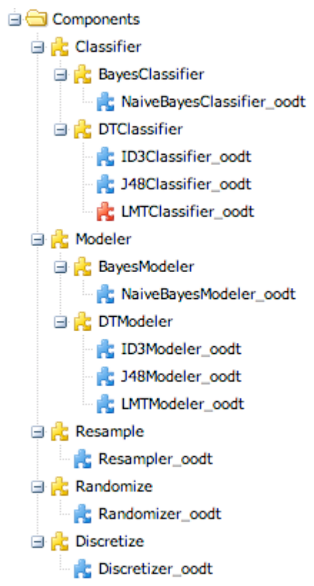
I. Mapping Semantic Workflows to Different Execution Engines Using WINGS

We have implemented this approach in the WINGS workflow system [6]. WINGS separates the specification of workflows at the domain layer and the specification of workflows at the logical layer. In WINGS, the ontologies for the CDoTs and CACs are represented using the W3C OWL standard. The workflow reasoning algorithms query these catalogs using SPARQL queries.

WINGS has been integrated with several execution engines. In this paper, we demonstrate our approach using two of them. OODT is a distributed data management framework, with components to extract metadata and do profiling, and with distributed execution of workflow components that can be managed from the workflow execution engine [9]. Pegasus/Condor is a workflow execution engine that selects distributed resources for the remote distributed execution of workflow tasks, optimizes the assignment of resources to tasks, and manages any data movements across resources [2].

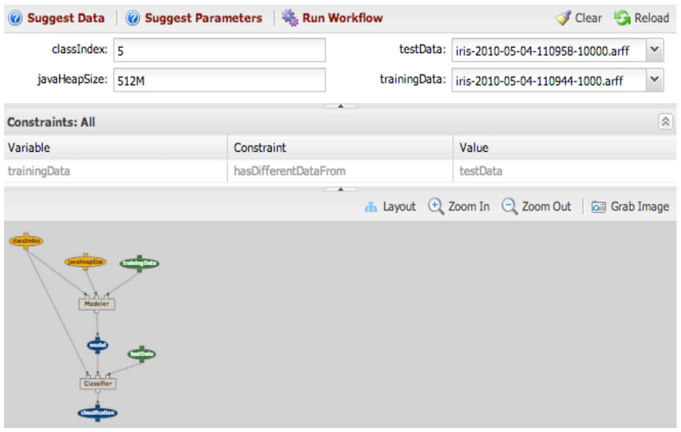
A. Catalogs of Domain Tasks in WINGS

Figure 2 shows a semantic workflow of domain tasks from a WINGS CDoT for a simple machine learning domain. This workflow takes training data and generates a model, then classifies any new test data according to that model.



- Components
  - Classifier
    - BayesClassifier
      - NaiveBayesClassifier_oodt
    - DTClassifier
      - ID3Classifier_oodt
      - J48Classifier_oodt
      - LMTClassifier_oodt
  - Modeler
    - BayesModeler
      - NaiveBayesModeler_oodt
    - DTModeler
      - ID3Modeler_oodt
      - J48Modeler_oodt
      - LMTModeler_oodt
  - Resample
    - Resampler_oodt
  - Randomize
    - Randomizer_oodt
  - Discretize
    - Discretizer_oodt

Figure 4. The CDoT components (shown in yellow) are mapped to the CAC components available in OODT for execution (shown in blue).



classIndex: 5
javaHeapSize: 512M
testData: iris-2010-05-04-110958-10000.arff
trainingData: iris-2010-05-04-110944-1000.arff

Constraints: All

Variable	Constraint	Value
trainingData	hasDifferentDataFrom	testData

Figure 5. Submitting the semantic workflow to the selected workflow execution engine.

B. *Catalogs of Application Codes in WINGS*

Figure 3 illustrates how a user selects a workflow execution framework in WINGS. In selecting OODT, the system imports the OODT CAC and maps it to the WINGS CDoT for this domain. Figure 4 shows the WINGS interface where the user can browse the CDoT in WINGS and see the mappings to the CAC in OODT. The domain tasks are shown as puzzle pieces organized in a hierarchy.

Figure 5 shows how a user submits a semantic workflow to the selected workflow execution engine. The user selects input datasets (for testData and TrainingData) and parameter values (for classIndex and javaHeapSize). Then the user selects "Run Workflow". WINGS uses workflow reasoning algorithms that take a semantic workflow of domain tasks and automatically generate workflows of executable application codes that can be submitted to a workflow execution engine.

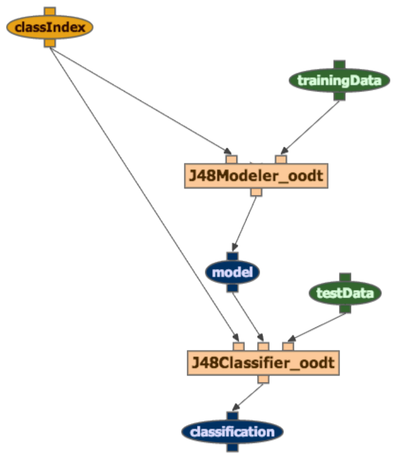


Figure 6. Workflow submitted to OODT for execution.

The workflow reasoning algorithms search through the space of possible specializations of the initial high-level workflows. To specialize a workflow, the algorithms query the CDoTs and the CACs to retrieve subclasses of a domain task class. The algorithms also query the catalogs regarding the properties of classes defined for domain tasks and application codes, which represent their arguments. There may be several possible executable workflows that are consistent with the initial domain-level workflow, since several application codes may be available for a given domain task. In that case, the user is presented with several options.

Figure 6 shows a workflow that WINGS can submit to OODT for execution. Note that the domain tasks that appeared in the workflow of Figure 2 have been automatically mapped to application codes available in the OODT execution environment and that appeared in the OODT CAC.

C. *Workflow Representations in P-PLAN*

WINGS uses the W3C PROV standard [5] to publish provenance of workflow executions. We have developed an extension of PROV called P-PLAN [4], which enables the representation of plans that mirror the execution structure (i.e., plans with no control constructs like conditionals or iterations). P-PLAN is used to represent the original workflows. WINGS uses P-PLAN to represent the workflow that is submitted to execution engines, such as the one shown in Figure 6. P-PLAN, in effect, provides a standard representation for workflows that WINGS uses to submit to all execution engines. Figure 7 illustrates how the semantic workflow in Figure 2 can be expressed in P-PLAN. The classes are highlighted in bold, relations (or properties) are in italics.

The representation of workflows is therefore in a language that is not specific to WINGS, facilitating reuse as well as querying of workflows from different workflow systems. In addition, WINGS has a facility to export both workflows of domain tasks and workflows of application codes in RDF as linked data [3]. This ensures that the workflow and all its constituents are web objects that can be openly accessed by third-party web applications.

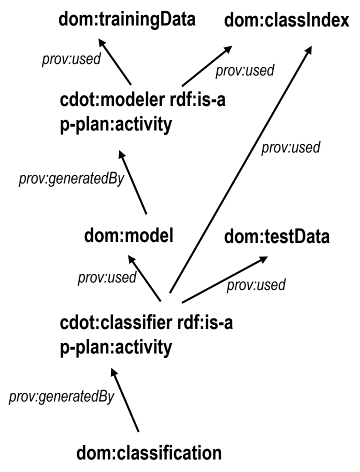
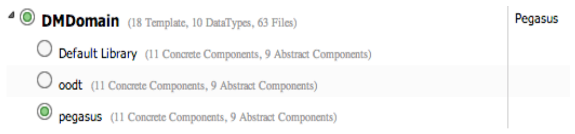


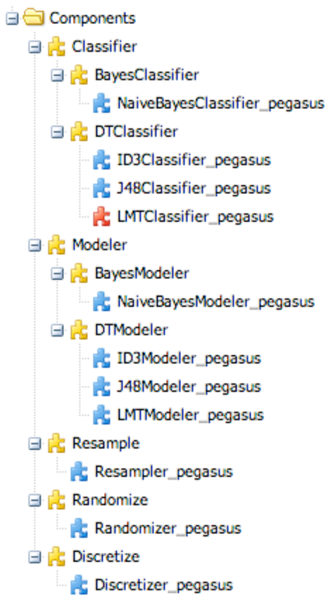
Figure 7. P-PLAN representation of the workflow in Figure 2.



- DMDomain (18 Template, 10 DataTypes, 63 Files)
  - ○ Default Library (11 Concrete Components, 9 Abstract Components)
  - ○ oodt (11 Concrete Components, 9 Abstract Components)
  - ● pegasus (11 Concrete Components, 9 Abstract Components)

Pegasus

Figure 8. Selecting Pegasus/Condor as the workflow execution environment.



- Components
  - Classifier
    - BayesClassifier
      - NaiveBayesClassifier_pegasus
    - DTClassifier
      - ID3Classifier_pegasus
      - J48Classifier_pegasus
      - LMTClassifier_pegasus
  - Modeler
    - BayesModeler
      - NaiveBayesModeler_pegasus
    - DTModeler
      - ID3Modeler_pegasus
      - J48Modeler_pegasus
      - LMTModeler_pegasus
  - Resample
    - Resampler_pegasus
  - Randomize
    - Randomizer_pegasus
  - Discretize
    - Discretizer_pegasus

Figure 9. The CAC defined in the Pegasus/Condor execution environment.

D. *Using Alternative Execution Engines*

It is easy to set up WINGS to run workflows in different execution engine, even in the same session. Figure 8 shows the selection of Pegasus/Condor as the workflow execution environment. Figure 9 shows the application codes available in the CAC of the Pegasus/Condor environment. Figure 10 shows the workflow generated by WINGS for submission to Pegasus/Condor for execution.

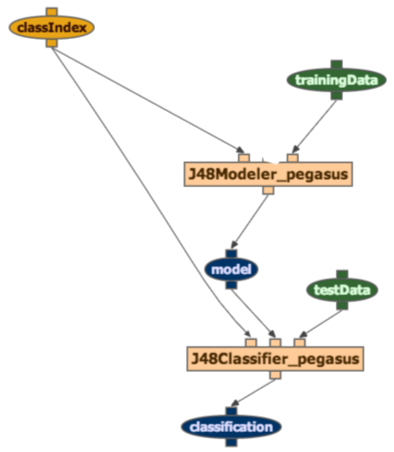
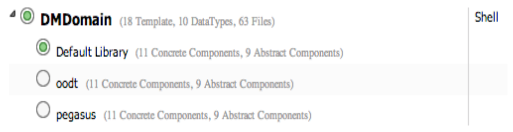


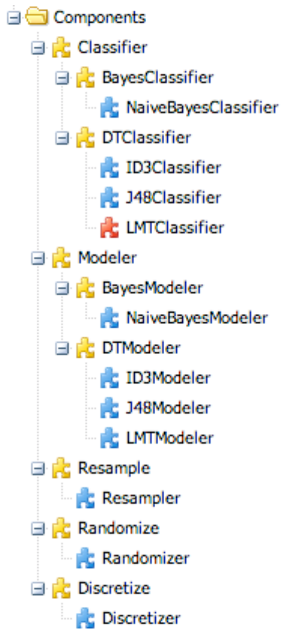
Figure 10. Workflow submitted to Pegasus/Condor for execution.



- DMDomain (18 Template, 10 DataTypes, 63 Files)
  - ● Default Library (11 Concrete Components, 9 Abstract Components)
  - ○ oodt (11 Concrete Components, 9 Abstract Components)
  - ○ pegasus (11 Concrete Components, 9 Abstract Components)

Shell

Figure 11. Selecting the local shell as the workflow execution environment.



- Components
  - Classifier
    - BayesClassifier
      - NaiveBayesClassifier
    - DTClassifier
      - ID3Classifier
      - J48Classifier
      - LMTClassifier
  - Modeler
    - BayesModeler
      - NaiveBayesModeler
    - DTModeler
      - ID3Modeler
      - J48Modeler
      - LMTModeler
  - Resample
    - Resampler
  - Randomize
    - Randomizer
  - Discretize
    - Discretizer

Figure 12. The CDoT and CAC mappings in the shell execution environment.

Finally, we show how the user can easily also submit the same semantic workflow for execution on a local shell. Figure 11 shows the switch to the local shell execution environment. Figure 12 shows the CDoT and CAC mappings. Figure 13 shows the workflow submitted.

Figure 14 shows the three executions of the semantic workflow that we showed in Figure 2. Here we have selected the second execution, run with OODT, and we show the OODT

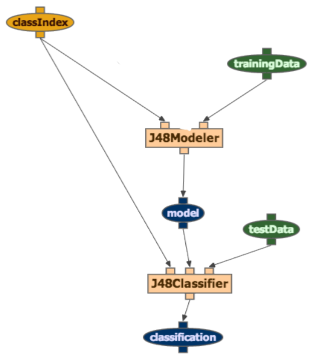
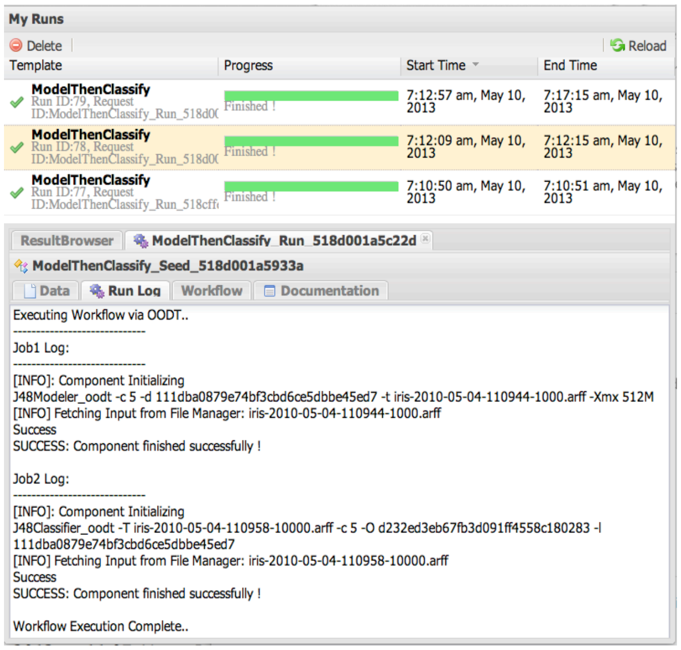


Figure 13. Workflow submitted to the shell for execution.



Template	Progress	Start Time	End Time
✓ ModelThenClassify Run ID:79, Request ID:ModelThenClassify_Run_518d0(	Finished !	7:12:57 am, May 10, 2013	7:17:15 am, May 10, 2013
✓ ModelThenClassify Run ID:78, Request ID:ModelThenClassify_Run_518d0(	Finished !	7:12:09 am, May 10, 2013	7:12:15 am, May 10, 2013
✓ ModelThenClassify Run ID:77, Request ID:ModelThenClassify_Run_518cff(	Finished !	7:10:50 am, May 10, 2013	7:10:51 am, May 10, 2013

ModelThenClassify_Run_518d001a5c22d

ModelThenClassify_Seed_518d001a5933a

```
Executing Workflow via OODT..
----------------------------
Job1 Log:
----------------------------
[INFO]: Component Initializing
J48Modeler_oodt -c 5 -d 111dba0879e74bf3cbd6ce5dbbe45ed7 -t iris-2010-05-04-110944-1000.arff -Xmx 512M
[INFO] Fetching Input from File Manager: iris-2010-05-04-110944-1000.arff
Success
SUCCESS: Component finished successfully !

Job2 Log:
----------------------------
[INFO]: Component Initializing
J48Classifier_oodt -T iris-2010-05-04-110958-10000.arff -c 5 -O d232ed3eb67fb3d091ff4558c180283 -l
111dba0879e74bf3cbd6ce5dbbe45ed7
[INFO] Fetching Input from File Manager: iris-2010-05-04-110958-10000.arff
Success
SUCCESS: Component finished successfully !

Workflow Execution Complete..
```

Figure 14. Records of the three executions of the same semantic workflow submitted from WINGS, each executed with a different workflow execution engine.

execution log. Note that the components used in each workflow execution framework are different, as shown in Figures 6, 10,and 13. The result browser shows those workflows in the "Workflow" tab. In each case, WINGS mapped the domain tasks in the semantic workflow into the application codes of the CAC available in the selected workflow execution system.

II. CONCLUSIONS

We have described an approach to representing semantic workflows of domain tasks. These workflows abstract the specification of steps from the particular application codes to be executed as well as the particular workflow execution engines used. We demonstrated this with two very different workflow execution engines, as well as execution on the shell.

Our work makes workflows more portable to new execution environments, where the availability of application codes and run-time libraries may vary. This also facilitates reproducibility and reuse. It also allows users to specify workflows at a domain level, which is closer to the descriptions of methods that are typically included in scientific articles.

An area of future work is that application codes with similar functionality may have different invocation signatures and parameters. In our current work, we assume that the invocation signatures are the same for all application codes that correspond to a domain task. We will need to extend the representations of the components, as well the workflow generation algorithm to take these differences into account.

ACKNOWLEDGMENTS

We thank the WINGS, Apache OODT, and Pegasus/Condor teams for their support of this work, in particular Varun Ratnakar and Chris Mattmann. We gratefully acknowledge support from the Air Force Office of Scientific Research (AFOSR) with award FA9550-11-1-0104, and from the Defense Advanced Research Projects Agency (DARPA) with award FA8750-13-C-0016.

REFERENCES

[1] De Roure, D; Goble, C.;Stevens, R. "The design and realization of the myExperiment Virtual Research Environment for social sharing of workflows". Future Generation Computer Systems, 25 (561-567), 2009.

[2] Deelman, E.; Singh, G.; Su, M.; Blythe, J.; Gil, Y.; Kesselman, C.; Kim, J.; Mehta, G.; Vahi, K.; Berriman, G. B.; Good, J.; Laity, A.; Jacob, J. C.; and Katz, D. S. "Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems." Scientific Programming, 13(3), 2005.

[3] Garijo, D., and Gil, Y. "A New Approach for Publishing Workflows: Abstractions, Standards, and Linked Data". Proceedings of the Sixth Workshop on Workflows in Support of Large-Scale Science (WORKS'11), held in conjunction with Supercomputing (SC), 2011.

[4] Garijo, D., and Gil, Y. " Augmenting PROV with Plans in P-PLAN: Scientific Processes as Linked Data". Proceedings of the Second International Workshop on Linked Science (LISC'12), held in conjunction with the International Semantic Web Conference (ISWC), 2012.

[5] Gil, Y. and S. Miles (Eds). "A Primer for the PROV Provenance Model." W3C Recommendation, April 2013.

[6] Gil, Y.; Gonzalez-Calero, P. A.; Kim, J.; Moody, J.; and Ratnakar, V. "A Semantic Framework for Automatic Generation of Computational Workflows Using Distributed Data and Component Catalogs." Journal of Experimental and Theoretical Artificial Intelligence, 23(4), 2011.

[7] Hauder, M., Gil, Y. and Y. Liu. "A Framework for Efficient Text Analytics through Automatic Configuration and Customization of Scientific Workflows", Proceedings of the Seventh IEEE International Conference on e-Science, Stockholm, Sweden, December 5-8, 2011.

[8] Kozlovszky, M., Karoczkai, K., Marton, I., Balasko, A., Marosi, A., Kacsuk, P. "Enabling Generic Distributed Computing Infrastructure Compatibility for Workflow Management Systems." Computer Science, 13(3), 2012.

[9] Mattmann, C.; Crichton, D.; Medvidovic, N.; and Hughes, S. "A Software Architecture-Based Framework for Highly Distributed and Data Intensive Scientific Applications." Proceedings of the 28th International Conference on Software Engineering (ICSE06), pp. 721-730, Shanghai,China, 2006.

[10] Oinn, T., M. Greenwood, M. Addis, N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. "Taverna: lessons in creating a workflow environment for the life sciences." Concurrency and Computation: Practice and Experience, 18(10), 2006.

[11] Taylor, I. J., Deelman, E., Gannon, D. B., and M. Shields (Editors), "Workflows for e-Science: Scientific Workflows for Grids," Springer, January 2007.