Integrating Existing Scientific Workflow Systems: The Kepler/Pegasus Example

Nandita Mandal, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, Karan Vahi

USC Information Sciences Institute

Marina Del Rey, CA 90292

{nandita, deelman, gmehta, mei, vahi}@isi.edu

ABSTRACT

Scientific workflows have become an important tool used by scientists to conduct large-scale analysis in distributed environments. Today there is a variety of workflow systems that provide an often disjoint set of capabilities and expose different workflow modeling semantics to the users. In this paper we examine the possibility of integrating two well-known workflow systems Kepler and Pegasus and examine the opportunities and challenges presented by such an integration. We illustrate the combined system on a workflow used as a basis of a provenance challenge.

Categories and Subject Descriptors

D.1 Programming Techniques.

General Terms

Design, Languages

Keywords

Scientific Workflows, Programming Models, User Interfaces

1. INTRODUCTION

Scientific workflows are quickly becoming recognized as an important unifying mechanism to combine scientific data management, analysis, simulation, and visualization tasks [1]. Scientific workflows often exhibit particular traits, e.g., they can be data-intensive, compute-intensive, or visualization-intensive, thus covering a wide range of applications from low-level "plumbing workflows" of interest to grid engineers, to high-level "knowledge discovery workflows" for scientists [2]. There are many workflow management systems today, each with their own strengths and weaknesses. When designing workflows, scientists need to choose their target workflow management system and in the process often need to tradeoff between the various capabilities. In this paper we examine the possibility of integrating two well-known management systems: Kepler [2] and Pegasus [3] in the hopes of leveraging their respective strengths. We describe our initial integration and show the results of our approach using an example workflow which formed the basis of the provenance challenge [4] which aimed at comparing and contrasting provenance models developed within a variety of system, most of which were workflow-based.

2. KEPLER

The Kepler scientific workflow system [2] provides domain scientists with an easy to-use system for capturing scientific workflows. Kepler attempts to streamline the workflow creation and execution process so that scientists can design, execute, monitor, re-run, and communicate analytical procedures repeatedly with minimal effort [5]. The system follows an actor-oriented modeling approach where individual workflow components (e.g., for data movement, database querying, job scheduling, remote execution etc.) are abstracted into a set of generic, reusable tasks. Instantiations of these common tasks can be functionally equivalent atomic components (called actors) or composite components (so-called composite actors or sub workflows) [6]. Figure 1 shows a snapshot of Kepler running a gene sequence workflow utilizing web services and data transformations.

Kepler's intuitive GUI (inherited from Ptolemy [7]) for design and execution, and its actor-oriented modeling paradigm make it a very versatile tool for workflow design, prototyping, execution, and reuse for both workflow engineers and end users. Kepler workflows can be exchanged in XML using Ptolemy's own Modeling Markup Language (MoML). [9]

3. PEGASUS

The Pegasus mapping and planning framework uses the concept of abstract workflows to describe and model abstract job computations in distributed environments, such as the grid.

The framework creates a separation between the application description and the actual execution. Users describe workflows in resource-independent ways and Pegasus maps them onto potentially multiple heterogeneous resources distributed across the wide area networks, while at the same time shielding the user from grid details [3]. Pegasus finds appropriate resources to execute the computations and modifies the user-specified workflow to execute on those resources. Pegasus also adds tasks for data

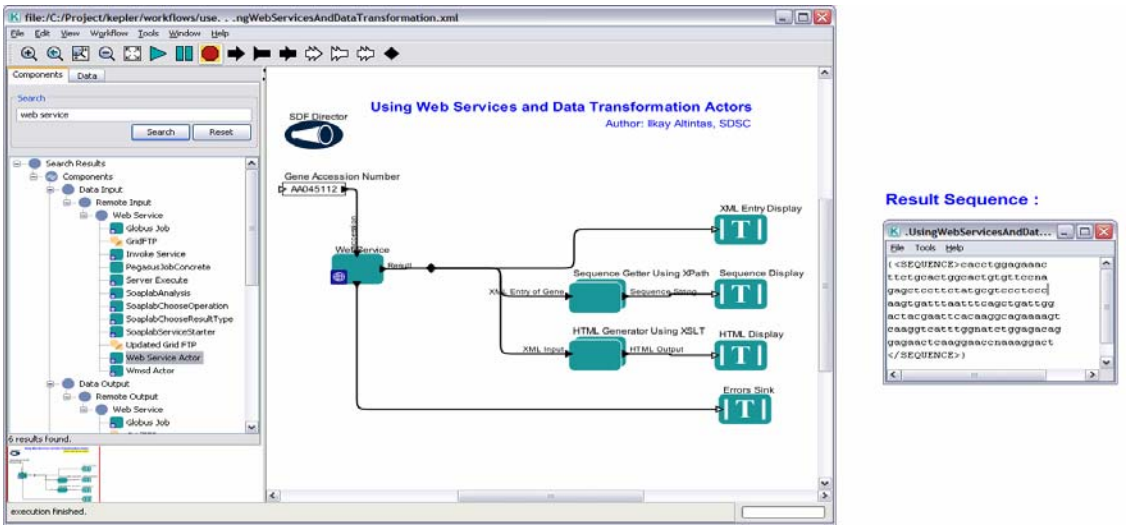

Figure 1. A bioinformatics workflow in Kepler which retrieves a gene sequence via web service and data transformation actors. The execution model is enforced by a director, SDF Director [8].

management by adding steps to the workflow to stage data to and from where the computations will take place, registering them into data registries and staging the results out to a user-specified location. Figure 2 gives a view of the above grid mapping process in Pegasus.

Currently Pegasus supports three main ways of specifying the abstract workflow: 1) using a semantic-rich workflow composition tool—Wings [10], using partial workflow descriptions—via VDL [11], and by directly specifying the workflow in an XML format [12] (DAX—Directed Acyclic Graph in XML). The abstract workflow is composed of tasks described in terms of logical transformations and logical input and output filenames. Pegasus consults monitoring services deployed in the environment to find the available resources. It also queries data registries to find the location of the data referred to in the workflow and queries the Transformation Catalog [13] to find the location of the workflow component executables. Based on all this information, Pegasus maps the workflow onto the resources and creates an executable workflow which is given to Condor DAGMan [14] for execution. DAGMan follows the dependencies described in the executable workflow and releases the tasks, as they become ready to run, to the execution environment.

4. INTEGRATED SYSTEM AND ITS BENEFITS

The goal of our integration is to provide the Kepler users with a system that allows them to develop workflows in a resource-independent way and thus obtain the benefits of workflow portability, optimization, and ease of design, and on the other hand, we want to provide the Pegasus users with a tool that allows a graphical method of workflow composition and a tool capable of visual workflow execution monitoring and debugging. Figure 3 shows the components of the integrated system.

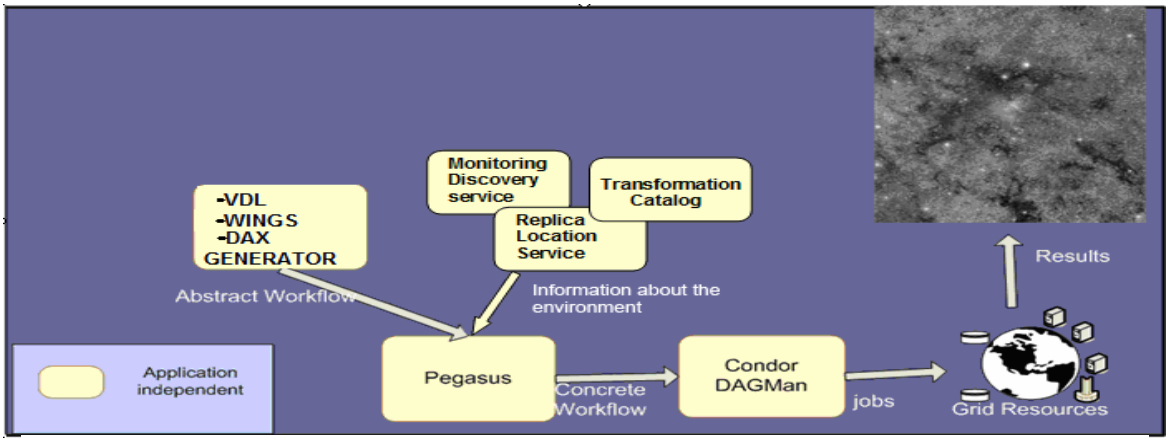

Figure 2. Pegasus converts abstract workflows into concrete workflows and maps them onto the grid.

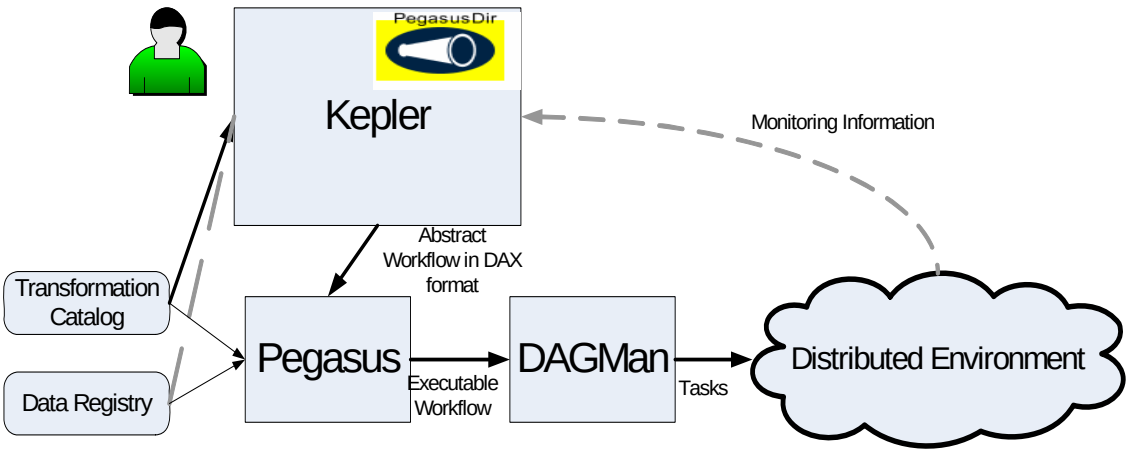

Figure 3: Integrated Kepler-Pegasus System. The light gray dotted lines indicate future work.

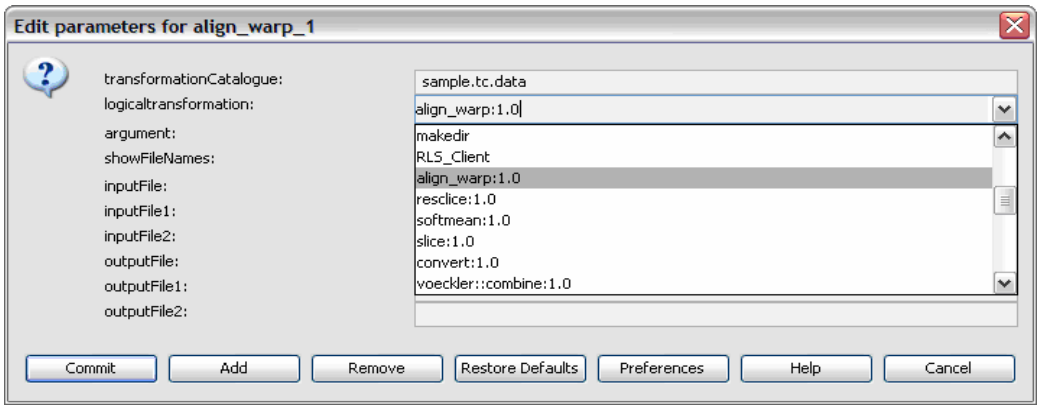

Figure 4a: User configuring the transformation for the Pegasus Job Abstract entity

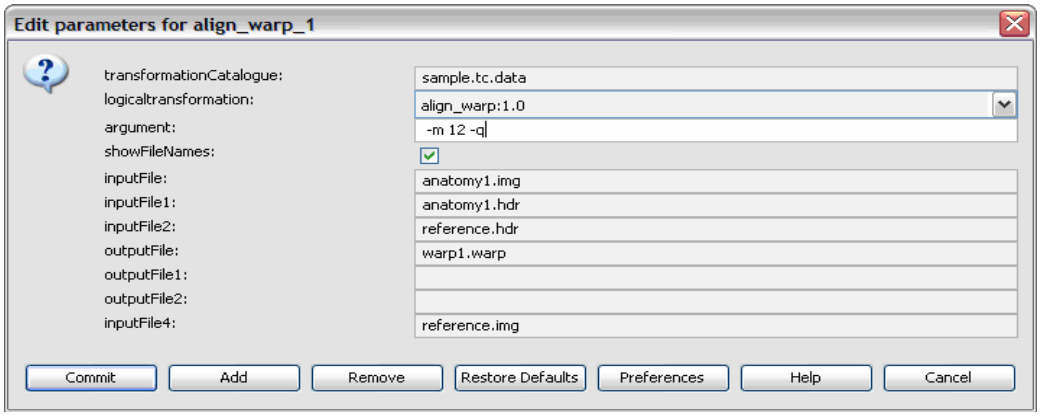

Figure 4b: User configuring the required command line arguments and I/O files for the ”align_warp” job.

The user composes the workflow graphically using the Kepler interface. The composition relies on the Transformation Catalog to find the available transformations. In the future we will also integrate the Data Registry into the workflow composition capability. Once the workflow is fully composed, the abstract workflow is created and given to Pegasus. The workflow mapping and execution then proceeds as described above with DAGMan managing the workflow execution in the distributed environment. Eventually, the monitoring information available in the environment will be presented to the user through the Kepler interface. Below we describe the workflow creation process in more detail.

The user interfaces with the system through the Kepler canvas and creates an abstract workflow in a visual form. This workflow refers to the actors (in Pegasus called transformations) and data files by their logical names. Figures 4a, and 4b show an example of a Pegasus actor's configuration dialog. The actor is generic and can take in multiple input files and can output multiple output files, hence contains multi-input/output port of width *n*. Since the actor is generic, the user needs to customize it to represent the desired computation to be performed within the workflow. The scientist is expected to input the logical transformation/computation he/she wants to perform on the grid. As seen in Figure 4.a a user enters a known Transformation Catalog (TC) to receive the list of available transformations. The Pegasus Transformation Catalog contains information about the location of the workflow components described in the abstract workflow as well as their resource requiremenst, the environment variables that need to be set, and any other information needed for the successful component execution on a remote resource. The interfaces in Figures 4a and 4b were customized specifically for Pegasus. Figure 4.b shows the user choosing appropriate parameters for the job. We discuss some of the issues associated with the generic actors in Section 2.2.

We implemented a Pegasus director for Kepler. The task of the director is to impose the dataflow model on the workflow designed by the user and to translate it into a DAX format suitable for Pegasus. This translation is made possible by utilizing the Kepler MoML [9] format consisting of actor entities with configuration and parameter information. The algorithm converts each entity in MOML to an appropriate job in the DAX format. Pegasus then maps this workflow onto the available resources and gives it to DAGMan for execution. In the fully integrated system, monitoring information would be flowing from the execution environment to Kepler. Currently this functionality is not implemented.

Next, we describe the integration points and the challenges we faced.

4.1 Different Execution Models

A workflow model defines the semantics of a workflow including its task and structure definitions. Kepler and Pegasus use two different types of workflow models: a concrete (executable) model and abstract model respectively. In the abstract model, a workflow is described in a resource-independent way. The abstract model provides a flexible way for users to define workflows without being concerned about low-level implementation details. Tasks in the abstract model are portable and can be mapped onto any suitable grid services at run-time by using resource discovery and mapping mechanisms. [15] The abstract model also eases the sharing of workflow descriptions between users within a community [14].

In contrast, the concrete model binds workflow tasks to specific resources and indicates specific data locations. In some cases, the concrete model may include nodes acting as data movement to stage data in and out of the computation and data publication to publish newly derived data into community data registries [14]. In another situation, tasks in the

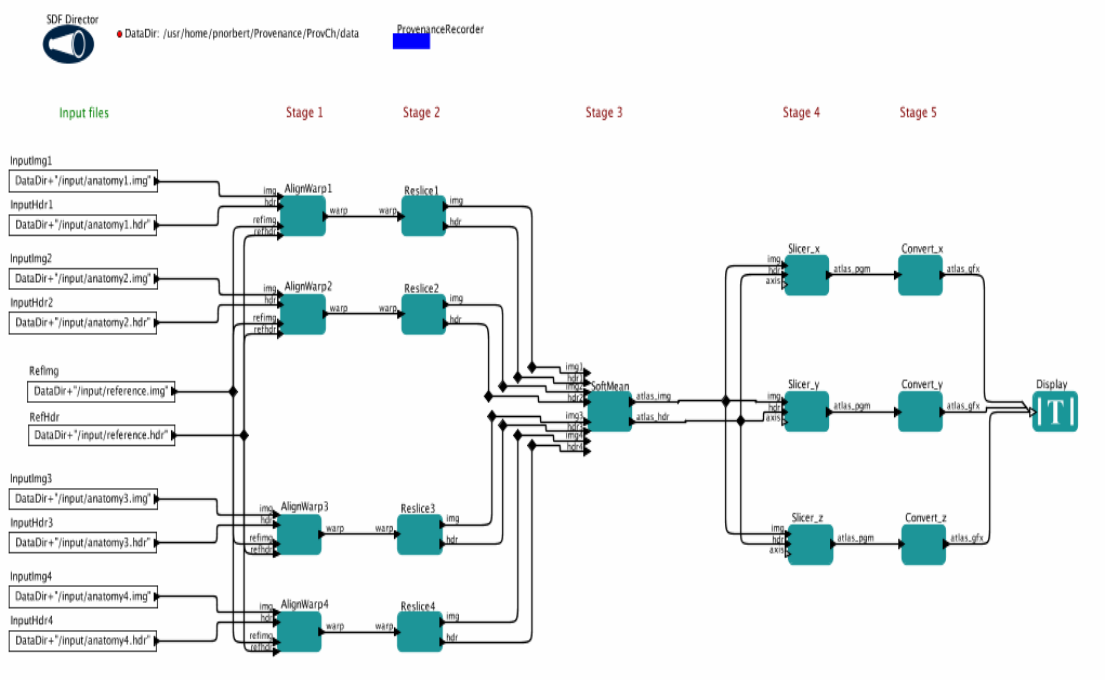

Figure 5: A Kepler concrete workflow for the Provenance Challenge FMRI imaging workflow [20]

concrete model may also include necessary application movement to transfer computational code Figure 5 shows an example of a concrete workflow designed in Kepler. This workflow was designed by the Kepler team in response to the Provenance Challenge [4] conducted in 2006. The challenge brought together researchers interested in comparing and contrasting the data provenance solutions developed within a variety of workflow and data management systems.

This example workflow was inspired by a real experiment in the area of Functional Magnetic Resonance Imaging (fMRI). This workflow was well documented to support the challenge and thus constitutes a good example that can be shared and explained to others. The jobs in the workflow create an averaged brain from a collection of high-resolution anatomical data and create 2D images across each sliced dimension of the brain. Inputs to the workflow are a set of 3D scans of brain images [4]. As is seen in the figure Kepler doesn't entertain the concept of generic grid jobs and the location of local input files (DataDir + "filename") is provided to each grid job by the user.

Figure 6 shows the same workflow implemented in our integrated system. The overall structure of the workflow remains the same, but now this workflow is abstract, in the sense that it is devoid of the execution details and thus is portable across execution environments. This workflow can be easily shared among collaborators to the data site for large-scale data analysis [15].

which have access to different resources. Another thing to notice is that the names of the files in the integrated system are unique within a workflow and can be also made unique across workflows. Pegasus uses the filename uniqueness to determine whether two files are the same and if they are Pegasus is able to optimize the workflow by not re-computing the data already available, if appropriate. Another thing to notice is that there are distinct names for each workflow component align_warp_1, align_warp_2,…, align_warp_4, although they refer to the same component and in a DAX format would have the underlying same name. This is because each component is a different job actor entity and it is not allowed to have same name for actors on Kepler canvas. This has ramifications for the Transformation Catalog as the catalog needs to have mappings for each actor rather than for each type of component.

4.2 Catalog Integration

As we have shown in Figure 4b, we have integrated the Pegasus Transformation Catalog with the Kepler environment in a way which allows the user to customize the Pegasus actor. Ideally, we would like to be able to create and customize the Pegasus actors on the fly so that the user would not have to go through the two-step process (add a Pegasus actor to the canvas and

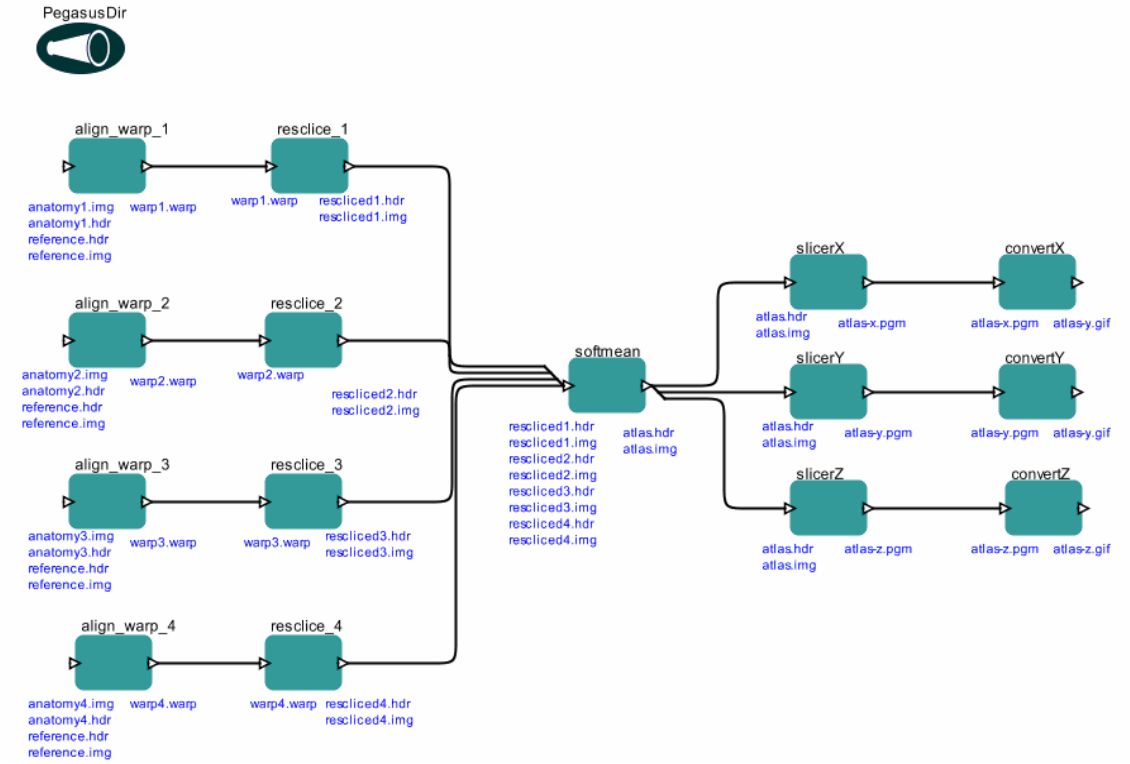

Figure 6: Abstract workflow created on the Kepler canvas with Pegasus Director and Pegasus Job Abstract entities.

then customize it to represent the desired computation) to create the desired workflow component. A solution is to view the workflow components as dynamic libraries on the Kepler canvas. Hence dynamic abstract jobs can be loaded as libraries in the Kepler environment. This would enable scientists to get a quick overview at all the logical transformations and jobs applicable and simply drag them on the canvas for further use. Figure 7 gives a snapshot of the Kepler library panel and the drag and drop nature of the actor entities. Kepler currently supports a library of all the concrete actors/entities that can be executed in non-abstract type workflows. As in the provenance challenge workflow an associated Transformation Catalog could contain all the operations of *align*, *reslice*, *softmean*, *slicer*, and *convert* job entites which can be simply loaded when the Kepler environment starts up. This dynamic loading of entities would require access of the most recent Transformation Catalog file so as to give an updated list of currently possible logical transformations. However, currently the Kepler library system does not allow dynamic loading of entities at start up and requires the creation of specific files for the actor entity to be registered with the framework.

The Kepler Object Manager is the infrastructure component designed to manage access to all objects on both the local filesystem and through network-accessible services. The managed objects including data objects, metadata objects and annotations, and actor classes, are identified using an LSID identifier. This identifier can be used to retrieve more detailed information about the component, including metadata about the component which will include a list of components on which this one depends.

Although it is not currently supported, the object manager could be modified to accept and automate the process of creating abstract actors from the logical transformation file as well as during start up process. This would require a sub component to create unique LSIDs for abstract logical transformations defined in the Transformation Catalog. A new semantic domain of "Pegasus" can be introduced to the environment and all the newly created abstract jobs can be registered as belonging to the "Pegasus" domain. This automated dynamic loading and registration of the abstract jobs would enable scientists to easily access grid jobs available for execution.

Another integration not currently in place is the integration of Kepler with the current Pegasus data registry. This registry is composed of a metadata catalog such as MCS [17] and a replication location service such as RLS [18]. The registry can be used by the scientists to discover data to input into the workflow. This integration could be done as a simple extension to the current integration of the Transformation Catalog with the Pegasus abstract actor. When the user chooses which files to use an input to a logical transformation, a data registry could be queried.

4.3 Monitoring and Debugging

After scientists have modeled and used the workflow tools to schedule their analysis for execution, the workflow management environment needs to provide monitoring information about the jobs currently executing on the grid. A specialized grid workflow monitoring tool can be implemented in the Kepler management system, where currently mapped workflows and jobs can be actively monitored. Users

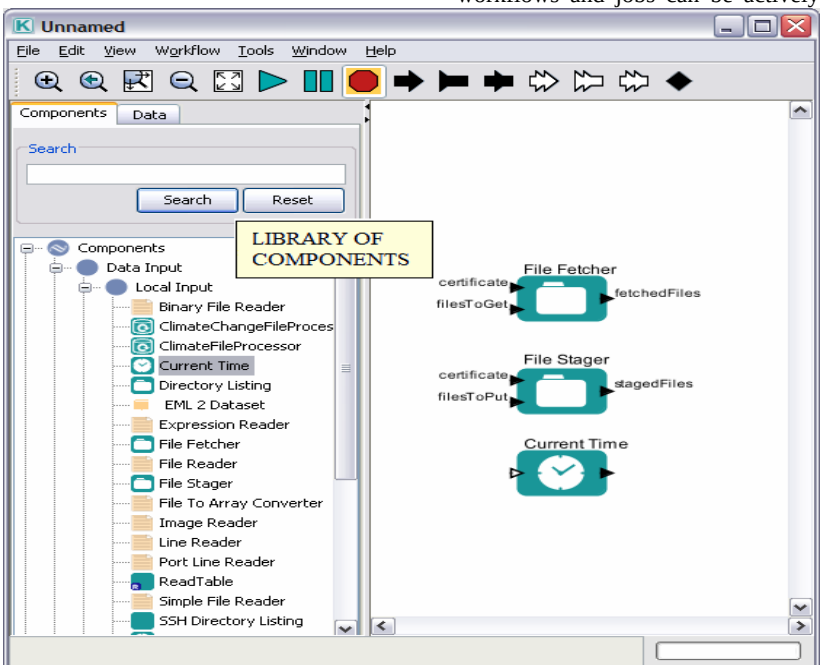

Figure 7: A snapshot of the Kepler environment with the left panel of library components which are loaded at runtime.

should be able to view the status of the workflow (submitted, active, done, failed), the number of tasks already completed, the tasks currently executing, and other information [16]. The accurate and up-to date job execution and status results details can be retrieved from a combination of the Metadata Catalog Service set up to keep track of active workflows, Condor-G, and DAGMan logs [14]. DAGMan creates a log file recording the execution history.

Many scientific abstract and concrete workflows which are mapped on the grid can consist of 100,000 jobs and more [22, 23]. Workflows with so many jobs cannot be manually created or visualized on the Kepler canvas. Even if loop constructs are used, the files and parameters needed cannot be easily visualized. Hence, Kepler can be first used as a modeling environment where scientists can initially create, test, and map small subsets of workflows and reproduce the above to the execution grid on a bigger scale. Additionally, the integrated environment can also be implemented as a debugging environment where workflow failures can be investigated. A scientist for example can debug a given abstract workflow. If the workflows had been mapped onto grid, then he can even inspect subsets of the concrete workflow versions of the above, to gain further understanding of the job execution and fault scenarios. Currently we have not implemented the monitoring and debugging capabilities.

4.4 Supporting current Pegasus users in the Integrated Environment

In order to support current Pegasus users, we developed a capability to import existing DAX workflows into Kepler and place them on the Kepler canvas for modification and visualization. However, these dynamically generated workflows require sophisticated graphing algorithms to better visualize the complex workflows. The integrated environment can make use of graph networks and visualization software such as “Grappa” [20] to better visualize complex nodes workflows consisting of 20-30 nodes or more in such an environment. We plan to support the use graph layout tools within Kepler in the future.

5. RELATED WORK

There are many systems today that are used by a variety of user communities [24, 25, 26, 27, 28, 29]. However, they often present different models and capabilities to the users. In the past there were some efforts to interface systems together. An example of a similar effort was done within the UK e-Science program under the “Link-Up” project [21]. This activity involved myGrid [19], Kepler [2], and the Wings/Pegasus system groups [30]. The goal was to combine workflow editors, validators, and generators from the different projects. This work is another step in this direction, taking two of the systems and integrating them in a way that allows users from both communities to benefit.

As we already mentioned in this paper there are also efforts in finding common ground in provenance systems as part of the Provenance Challenge [4].

6. CONCLUSIONS

In this paper we described a system that integrates the visual workflow composition aspects of Kepler and the workflow mapping and management capabilities of Pegasus. We believe that such an integrated system could benefit scientists in both user communities. Although we have shown that Kepler users can generate Pegasus-type abstract workflows in the Kepler environment and that current Pegasus users can use that environment to explore their workflows, much more work needs to be done. Currently Pegasus-type workflows generated in Kepler consist of custom Pegasus Job Abstract actors and directors. Further work to extend our system to fully incorporate all existing Kepler workflows and actors. We touched upon some of the issues related to workflow execution monitoring, where information from the execution environment needs to be propagated back to the user. We also described some of the issues associated with debugging, some of which are related to providing a mapping between the abstract and executable levels and some related to the scale of the workflows. In general, we believe that as the needs of the scientific communities grow, workflow management systems will have to learn to leverage the best of each other’s capabilities to deliver the tools that scientists need to do their work.

ACKNOWLEDGEMENTS

This work was supported by the National Science Foundation under grant OCI-0438712. We would like to thank Ilkay Altintas for helpful discussions.

REFERENCES

[1] Shawn Bowers et al., *Actor Oriented Design for Scientific Workflows*, Lecture Notes in Computer Science, Vol. 3716 (November 2005), pp. 369-384.

[2] Betram Ludascher et al., *Scientific Workflow Management and the Kepler System*. Concurrency and Computation: Practice and Experience, Special Issue on Scientific Workflows, 2005.

[3] Ewa Deelman et al., *"Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems"*. Scientific Programming Journal, Vol 13(3), 2005, Pages 219-237

[4] S. Miles, *First Provenance Challenge*, http://twiki.ipaw.info/bin/view/Challenge /FirstProvenanceChallenge, August 2006.

[5] Ilkay Altintas et al., *Kepler: An Extensible System for Design and Execution of Scientific Workflows*, 2004.

[6] Ilkay Altintas et al., *A Framework for the Design and Reuse of Grid Workflows*, Spring 2005.

[7] PTOLEMY II project and system. Department of EECS, UC Berkeley, 2004. http://ptolemy.eecs.berkeley.edu/ptolemyII/.

[8] C. Brooks et al., *Heterogeneous Concurrent Modeling and Design in Java (Volume 3: Ptolemy II Domains): SDF Domain*, Technical Memorandum UCB/ERL M04/17, University of California, Berkeley, CA USA 94720, June 24, 2004.

[9] Edward A. Lee and Steve Neuendorffer. *MoML — A Modeling Markup Language in XML — Version 0.4*. Technical report, University of California at Berkeley, March, 2000.

[10] Yolanda Gil et al., *Wings for Pegasus: A Semantic Approach to Creating Very Large Scientific Workflows*, OWL: Experiences and Directions 2006

[11] I. Foster et al., *Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation*, Proceedings of Scientific and Statistical Database Management, 2002.

[12] Ewa Deelman et al., Pegasus*: Mapping Scientific Workflows onto the Grid*, Across Grids Conference 2004, Nicosia, Cyprus

[13] E. Deelman, C. Kesselman, et al., "Transformation Catalog Design for GriPhyN," Technical Report GriPhyN-2001-17, 2001.

[14] Condor Team, *The directed acyclic graph manager*, *w*ww.cs.wisc.edu/condor/dagman, 2002

[15] Jia Yu and Rajkumar Buyya, *A Taxonomy of Workflow anagement Systems for Grid Computing*, Technical Report, GRIDS-TR-2005-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, March 10, 2005.

[16] Gurmeet Singh et al., *The Pegasus Portal: Web Based Grid Computing*, Symposium on Applied Computing archive Proceedings of the 2005.

[17] Gurmeet Singh et al., *A Metadata Catalog Service for Data Intensive Applications*, ACM/IEEE conference on Supercomputing 2003.

[18] A. Chervenak, E. Deelman, et al., *Giggle: A Framework for Constructing Scalable Replica Location Services*, Proceedings of Supercomputing 2002 (SC2002), Baltimore, MD. 2002.

[19] Carole Goble, *Using the Semantic Web for e-Science: Inspiration, Incubation, Irritation* Lecture Notes in Computer Science 3729:1-3

[20] J. Mocenigo, *Grappa: A Java Graph Package*, October 2006.

[21] http://www.mygrid.org.uk/linkup

[22] Berriman, G. et al., *Montage: The Architecture and Scientific Applications of a National Virtual Observatory Service for Computing Astronomical Image Mosaics*, Proceedings of Earth Sciences Technology Conference, 2006

[23] Ewa Deelman et al., *Managing Large-Scale Workflow Execution from Resource Provisioning to Provenance tracking: The CyberShake Example*, e-Science 2006, Amsterdam, December 4-6, 2006

[24] S. McGough et al., *Workflow Enactment in ICENI.* In UK e-Science All Hands Meeting, Nottingham, UK, IOP Publishing Ltd, Bristol, UK, Sep. 2004; 894-900.

[25] F. Berman et al., *The GrADS Project: Software Support for High-Level Grid Application Development*. International Journal of High Performance Computing Applications*(JHPCA)*, 15(4):327-344, SAGE Publications Inc., London, UK, Winter 2001.

[26] G. von Laszewski, M. Hategan. *Java CoG Kit Karajan/GridAnt Workflow Guide*. Technical Report, Argonne National Laboratory, Argonne, IL, USA, 2005.

[27] G. von Laszewski, K. Amin, M. Hategan, N. J. Zaluzec, S. Hampton, and A. Rossi. *GridAnt: A Client-Controllable Grid Workflow System.* In 37th Annual Hawaii International Conference on System Sciences *(HICSS'04)*, Big Island, Hawaii: IEEE CS Press, Los Alamitos, CA, USA, January 5-8, 2004.

[28] I. Taylor, M. Shields, and I. Wang. Resource Management of Triana P2P Services. *Grid Resource Management*, Kluwer, Netherlands, June 2003.

[29] T. Oinn et al., *Taverna: a tool for the composition and enactment of bioinformatics workflows.*Bioinformatics, 20(17):3045-3054, Oxford University Press, London, UK, 2004.

[30] Y. Gil, et al., "Wings for Pegasus: Creating Large-Scale Scientific Applications Using Semantic Representations of Computational Workflows," in Proceedings of the 19th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI) Vancouver, British Columbia, Canada, 2007 (to appear)