On the Black Art of Designing Computational Workflows

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

Pedro A. González-Calero
Facultad de Informática
Universidad Complutense de Madrid
28040 Madrid, Spain
+34 91 394 7517
pedro@sip.ucm.es

Ewa Deelman
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey CA 90292
+1 310 822 1511
deelman@isi.edu

ABSTRACT
Computational workflows have recently emerged as an effective paradigm to manage large-scale distributed scientific computations. Workflow systems can automate many execution-level details and provide assistance in composing and validating workflows. However, there is still a significant effort involved in creating these workflows since they often represent collaborative and exploratory science experiments. Therefore, current practice is effective in producing results but not cost-effective for widespread adoption. Drawing from our previous research in computational workflows across scientific disciplines, this paper analyzes the tasks and overall process for designing these workflows. We discuss software engineering methodologies and their relevance to creating workflows as a unique kind of software artifact. We also discuss our ongoing work to make workflow applications more cost effective and lower the barriers for widespread adoption of workflow technologies.

Categories and Subject Descriptors
C. Computer systems organization, D.2 Software engineering, D.2.10 Design.

General Terms
Design, Performance, Human Factors

Keywords
Scientific workflows, computational workflows, software design, workflow design, workflow systems

1. INTRODUCTION
Workflows have recently emerged as an effective paradigm to manage large-scope terascale scientific analyses and are a crucial technology to scale up to petascale levels [1, 2]. Existing workflow systems [3, 4, 5] have been demonstrated in a variety of scientific applications where workflow creation draws from catalogs of hundreds of distributed software components and data sources, where the generation of workflows of thousands of interrelated computing processes is largely automated, and where the execution of workflows takes place on high-end computing resources and often spans several months.

The focus of our work and of this paper is on *computational workflows* [7]. Computational workflows are composed of portable codes that can be submitted for execution to several alternative execution resources (ranging from single-host to cluster platforms), process large-scale datasets, and can be easily restructured to exploit parallel data processing. In contrast, other applications use service-based workflows that are assembled from existing services [4, 5] that are often managed by third parties. Our group has worked with a number of scientific communities to develop workflows for a number of large-scale software systems: astronomy, earthquake science, gravitational-wave physics, neuroscience, data mining, natural language processing, and others [3, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]. Our Wings/Pegasus workflow system can automate some aspects of workflow generation and execution and can also assist users in workflow creation and validation.

Creating these workflow applications is a complex and costly process in itself. They mobilize entire research communities to design these complex applications in collaborative efforts spanning often several months. The workflow creation process involves selecting appropriate data, integrating several models into an end-to-end computation, and most likely reformatting data between computations. Additional complexity stems from the execution environment where the necessary resources are often geographically distributed and data of interest may reside on multiple storage systems. The cost of developing a workflow application is high and it is often only done for high payoff scientific quests. Alas, current practice is not a recipe that is cost-effective for even medium-payoff quests, for scaling up and accelerating the pace of science, or to be adopted by other disciplines that may not be able to afford making such investments.

Therefore, in order for computational workflows to become more commonplace in scientific practice, it is crucial to understand the current process of creating computational workflows so it can be improved and made more efficient. By analyzing the process, we

WORKS'07, June 25, 2007, Monterey, California, USA.

can find opportunities to assist or automate parts of the process that contribute to the cost of developing workflow applications.

In our work, we have found that one of the most important questions that a prospective user community asks is what does it take to create a workflow-based application: how is it done, and what is the cost. We describe to them the different stages involved in the process of designing and creating workflows, estimate the effort and cost of each stage for their particular application, and highlight the benefits of workflow-based approaches that will make the investment worthy. In some cases, the application is developed from scratch, where the design of the workflow and the design of the scientific experiment and analysis go hand in hand. In other cases, the starting point is an existing body of software, often including many modules whose execution is either orchestrated by hand or through scripting languages. In those cases, moving to a workflow framework may have clear benefits for reasons of scale or usability, but the cost involved may be an obstacle to adoption. Therefore, it is important to understand the sources of cost and investigate whether current workflow systems can be extended to assist the process and reduce the effort required.

This paper gives an overview of our workflow creation process and summarizes best practices and lessons learned from our group's experiences over the years working with a variety of applications and communities. We continue to improve the process and to extend the capabilities of the workflow system as new applications bring in new requirements. This paper should be of interest to prospective user communities that are considering using a workflow system for scaling up an application. It should also be of interest to other workflow researchers that are running into similar issues and challenges in developing a workflow design process and cost-benefit estimates. This paper should also be of interest to software engineers as it illustrates the distinct process of creating workflows as a unique kind of software artifact.

The paper begins describing our approach to workflow creation by adding different kinds of information in successive layers, largely automating lower-level layers while providing assistance to users in higher-level layers that are domain-relevant. Next, we describe the process of workflow design and the kinds of effort involved at each step. Finally, we discuss the relationship of this work in the context of software engineering methodologies, and finalize with our planned future work in extending our workflow system to support the workflow design process.

2. LAYERED WORKFLOW DESCRIPTIONS

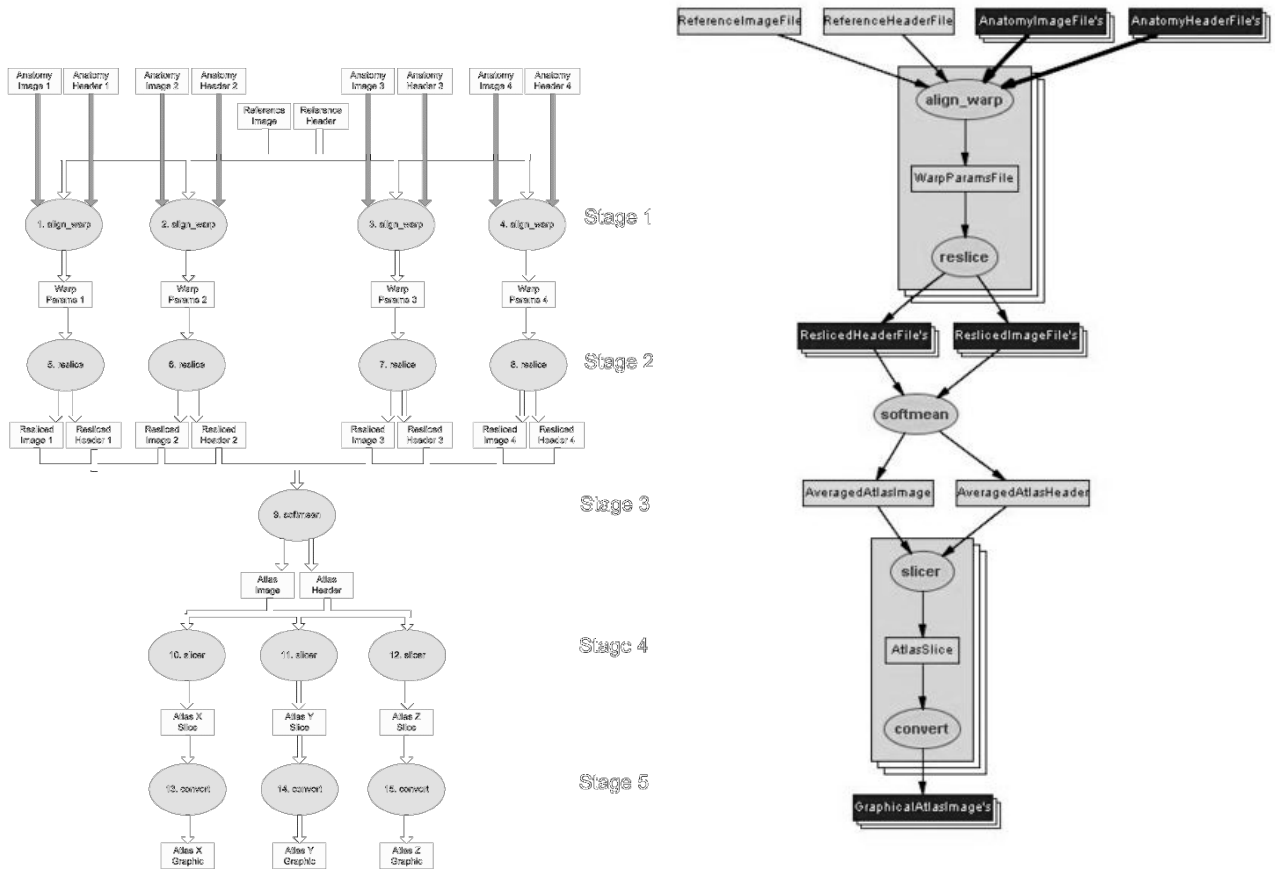
Representing workflows at appropriate levels of abstraction is key to managing the complexity of creating workflows and enables workflow systems to automate parts of the creation process and assist with others [7, 18].

We consider four layers of workflow descriptions:

1. **Workflow sketches**, which are informal (and often graphical) descriptions of computations and their dataflow. The workflow sketch is very low cost to produce, and provides a good description of the nature of the workflow to be created. It is also useful as a high-level roadmap of the work to be done.
2. **Workflow templates**, which are data- and execution-independent specifications of computations. Workflow templates identify the types of components to be invoked and the data flow among them. The nature of the components constrains the type of data that the workflow is designed to process, but the specific data to be used are not described in the template. In this sense, a workflow template is parameterized where its variables are data holders that will be bound to specific data in later stages of the workflow creation process. A workflow template can be shared and reused among users performing the same type of analysis.
3. **Workflow instances**, which specify the input data needed for an analysis in addition to the application components to be used and the data flow among them. They are execution-independent and still at the level of the application domain. A workflow instance can be created by selecting a workflow template that describes the desired type of analysis and by binding its generic data descriptions to specific data to be used. While a workflow instance logically identifies the full analysis, it does not include execution details such as the physical replicas or locations to be used. That is, the same workflow instance can be mapped into different executable workflows that generate the same results but use different resources available in alternative execution environments.
4. **Executable workflows**, which assign actual resources in the execution environment to all the computations specified in the workflow instance. Executable workflows fully specify the resources (e.g., physical replicas, sites and hosts, and service instances) that should be used for execution. This mapping process can be automated and ideally is incremental and dynamic in response to failures and other changes in the execution environment.

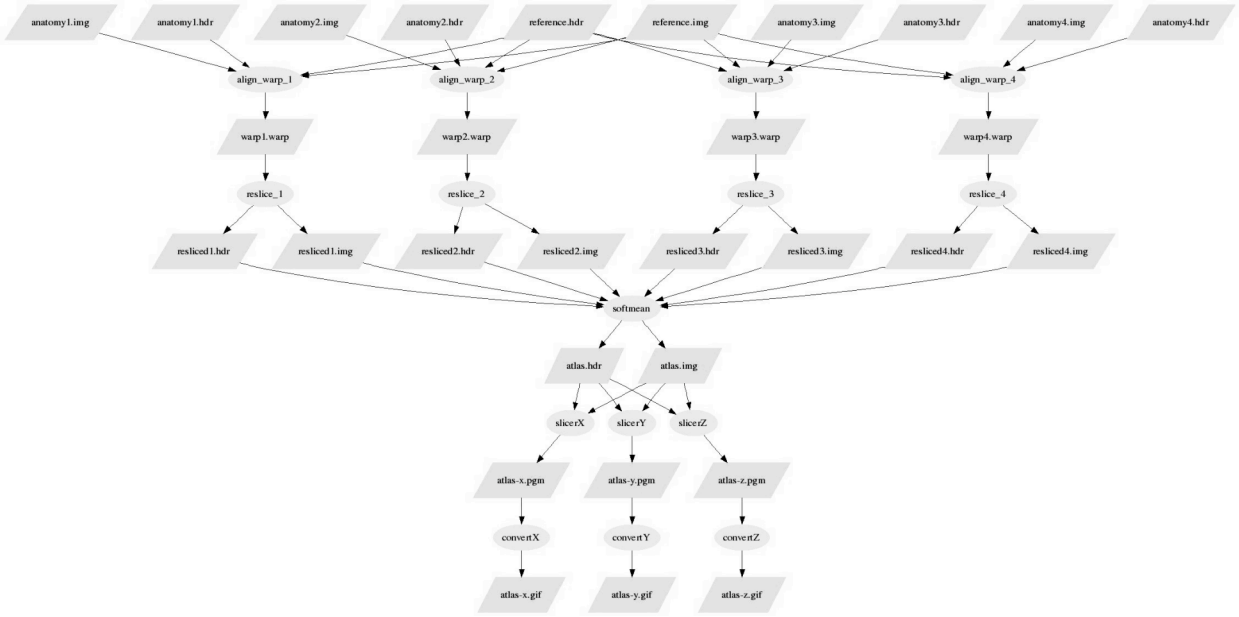
Most scientific workflows are created directly at the layer of executable workflows or workflow instances, making the creation process highly manual and therefore costly, prone to error, and not scalable. Workflow systems can automate the creation of workflows at the lower layers, and can assist users with component and data selection at domain-relevant layers [1]. The terms "abstract workflow" and "concrete workflow" are sometimes used to refer to a workflow instance and an executable workflow. We do not use the terms abstract and concrete in our work, since all our layers are abstractions of different types of information in workflows.

Figures 1 and 2 illustrate these four layers in an image processing workflow taken from [6], all the details and the four layers of representations of this workflow in our workflow system are described in [19] and are available on-line at http://vtcpc.isi.edu/provenance/index.php/Main_Page. Figure 1(a) shows a workflow sketch, which depicts graphically five stages of processing. The first two stages do warp aligning and reslicing on the initial set of images. All the images are processed through a single step in the third stage, and then sliced and converted in the fourth and fifth stages respectively. The workflow template shown in Figure 1(b) is a formal model of that sketch, notice that data collections are marked in dark overlayed rectangles and multiple parallel computations are grouped and marked in grey overlayed rectangles. Figure 2(a) shows the automatically expanded workflow instance that specifies individual data processing jobs, and 2(b) the executable workflow that includes



(a) workflow sketch (b) workflow template

Figure 1: In our layered workflow design approach, users design a *workflow sketch* and from it create a formal model of the components and their data flow in a *workflow template*.



(a) workflow instance

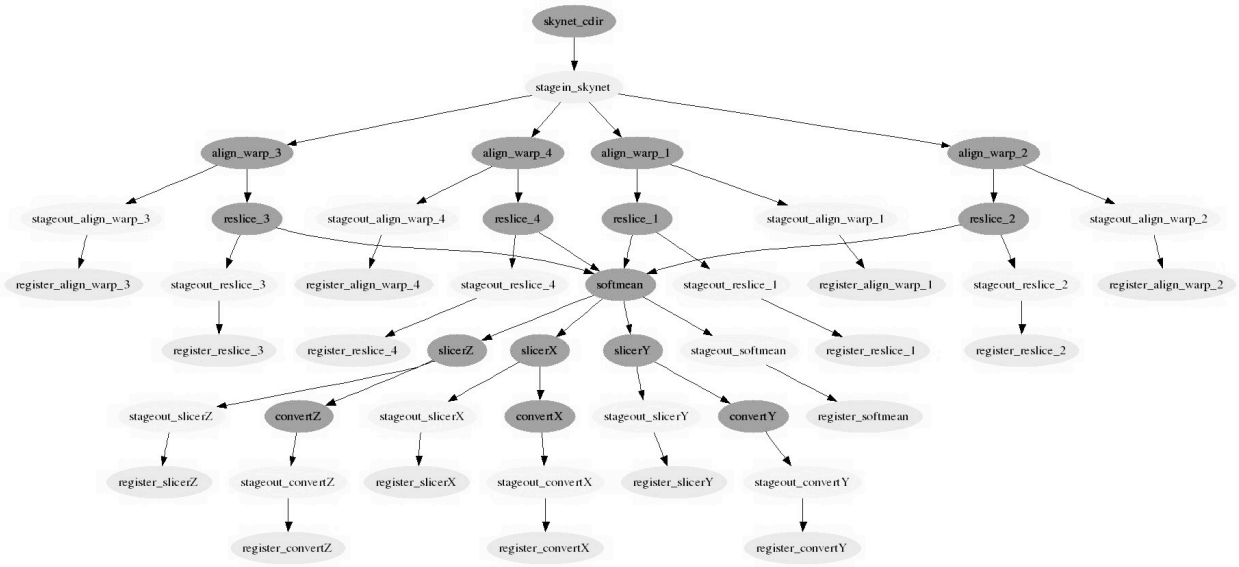


Figure 2: The workflow system automatically creates the workflow instance, and from that creates the executable workflow.

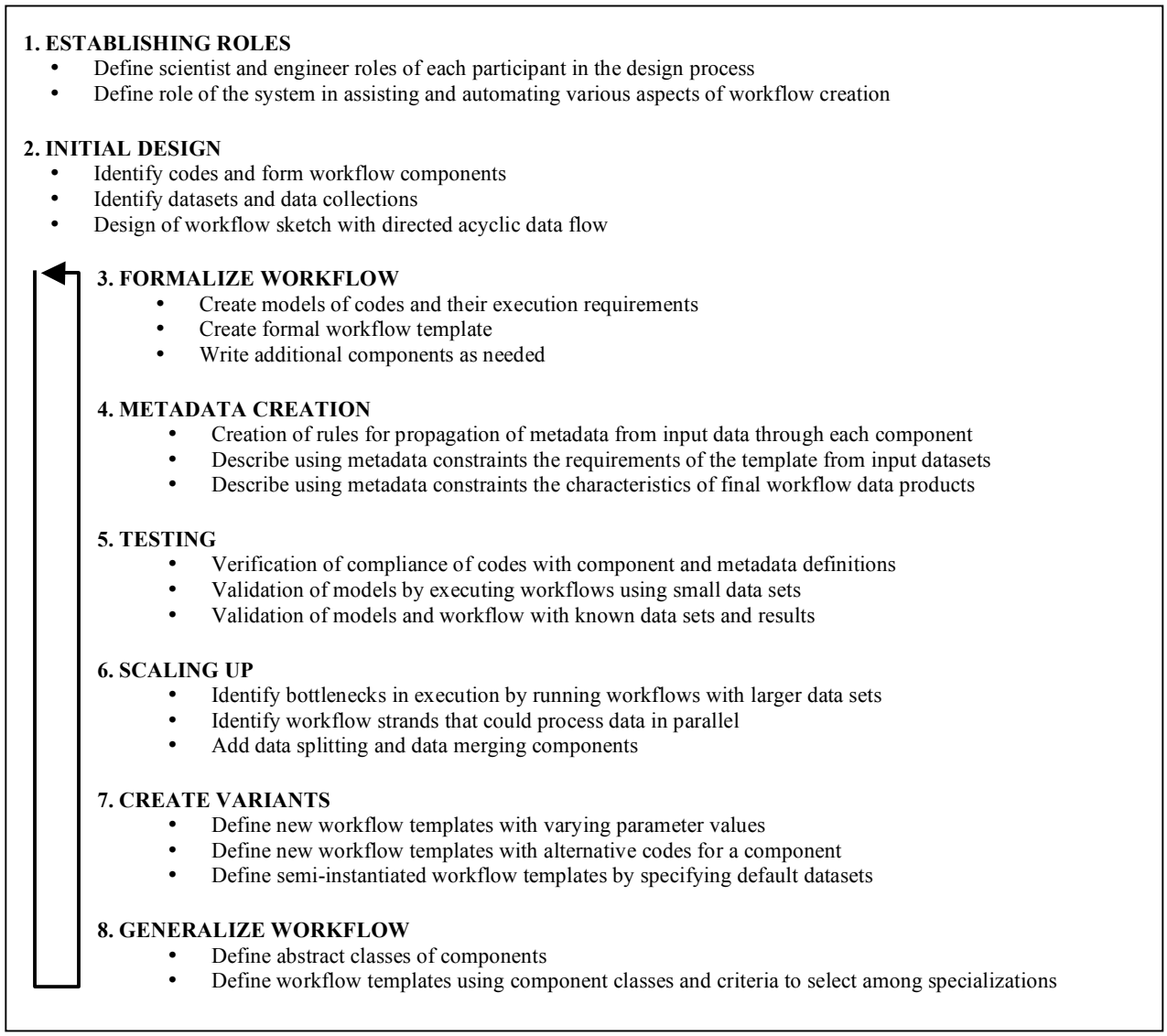


Figure 3: Our Current Process of Workflow Design.

automatically added data movements between catalogs and execution sites.

The next section describes how each of these four layers of workflow descriptions are generated at different stages of the workflow creation process.

3. THE PROCESS OF DESIGNING WORKFLOWS

This section describes the current process that we follow in designing workflows. We describe each stage of the process, discussing the best practices and lessons learned from our experiences.

Figure 3 illustrates the overall workflow design process. It consists of two initial steps and a non-specified number of iterations through the rest of the steps. Typically first iterations will provide a growing number of fully specified workflow templates along with workflow instances, executable workflows and information about workflow execution, while in later iterations more abstract workflows should be produced. The models obtained in different steps of the process would typically evolve as iterations proceed and a better understanding of the domain is built.

3.1 Establishing Roles

The process of computational workflow design is highly collaborative and cross-disciplinary in nature. A typical application may involve several dozen people including scientists, software engineers, code developers, workflow representation experts, and workflow execution experts. Scientists need to learn how to cast an analysis as a workflow. Software engineers and code developers need to learn how to turn codes into appropriate

workflow components so they can be automatically and remotely executed by the workflow system, and how to integrate them into the end-to-end workflow. The workflow representation experts need to learn how the application and the codes are designed to work together. The workflow execution experts need to learn the code execution requirements and failure conditions.

As the virtual collaboration is starting and the roles and abilities of each participant are being dynamically established, there is a lot of communication and concurrent design of different aspects of the workflow, and the right information needs to be conveyed by the appropriate parties at each point during the process.

Establishing the role and functionality of the workflow system is perhaps the most important aspect at this stage. Because workflows are often assembled from existing codes that implement a portion or a simpler version of the application envisioned, many of the functions that the workflow system can automate were originally automated in some way in the codes and in the scripts that glue the codes. For example, data movements, data depositions, metadata generation, and resource assignments for execution are all done automatically by the workflow system that we use. These functions must be well understood so that the codes can be turned into appropriate workflow components.

3.2 Initial Design

This stage involves identifying workflow components and data and understanding their dependencies in the computations. A workflow sketch is simply a diagram of the components and data flow in the workflow. Often, domain experts create sketches that include inadequate components for processes that are either automatically handled by the workflow system or would not support distributed execution appropriately. These include components for script-based control, data movement, and data and metadata management. For components of the workflow that will run existing code, care must be taken to remove hard-coded control flow through global variables and data manipulated by the codes.

The data flow also must be expressed in the workflow, so that components expose data passing done through local file systems into workflow data flow.

The codes themselves have to be cleaned up so they can report execution failures with appropriate error conditions in order for the workflow system to handle failure situations. Many of these issues often result from the kind of code base that is to be used for workflow components. Scripts may have been created to integrate running codes and manage their execution, which often lead to ad-hoc and inflexible solutions.

This stage of the process typically involves: domain experts (scientists), who provide the understanding of the available components and their interactions; workflow representation experts, who validate the workflow sketch; workflow execution experts, who identify requirements related to the workflow execution environment; and code developers, who modify the available components to fulfill workflow system requirements.

3.3 Formalize Workflow

A workflow template in our approach is a formal representation of the workflow sketch, including all the components required to perform the computations that the workflow is intended to make. The formal description of the workflow is built from the formal descriptions of its components which must be obtained also at this stage.

Although software component modeling is a hard problem, at this point we only need a shallow model where input/output data type constraints are represented. Those constraints serve to guarantee the correctness of the data flow in the workflow and also allow identifying possible gaps in that flow. The formal representation of the components and their connections may uncover the existence of gaps in the workflow that require new components to be created. These initial shallow models can be further elaborated at the next step in the workflow process for metadata creation.

The component models also include execution requirements, including the type of architecture needed, software installations including operating system and other software libraries, and minimum memory required.

This process involves: domain experts (scientists) to provide component descriptions, validate the workflow templates and propose components to fill the gaps; workflow representation experts to build the formal representation of the components and of the workflow templates; and code developers to write new components when needed.

3.4 Metadata Creation

Metadata serves different purposes in our approach. It serves to identify datasets that can be reused from previous computations and need not be computed again. Metadata also describes data characteristics that allow selecting a workflow based on the characteristics of its data products, and also let us select actual inputs given a workflow input requirements. And, finally, metadata represents semantic constraints between different pieces of data used to instantiate a workflow template, such as constraints on input *a* given input *b*, constraints on output given input and vice versa.

The most challenging task in this process is that of eliciting knowledge that the experts may have not ever before explicitly stated. Scientists must provide their expertise on the problem domain while software engineers provide their knowledge on the semantic constraints of the component implementation, and workflow representation experts formally represent them all. As new workflows and new metadata are created, metadata can be validated against the workflows obtained in previous iterations of the workflow creation process.

3.5 Testing

In this stage every new workflow template is validated through execution. Several workflow instances can be built from the workflow template in order to test different aspects of the computation.

First an instance is built using small datasets in order to test the functionality of the workflow: every piece of code connects properly and computes what it is supposed to. Then, several instances, preferring again simple data sets, can be used to test metadata constraints and validate the model of metadata propagation defined in the previous step. Finally, the workflow is used to solve actual problems and its performance is compared to the results obtained by the former non workflow-based application or by other means. In this stage, failure conditions of the code are detected and refined.

The testing stage can benefit from the existence of benchmark data sets built to reflect different aspects of the problem domain, as identified by the scientists, and special cases of the algorithms used by the components, as defined by the software engineers and code developers. Workflow representation experts validate the models while execution experts run the workflows.

3.6 Scaling Up

Workflow templates can be further refined in order to take advantage of the grid-enabled workflow execution system, by identifying workflow strands that could process data in parallel.

Parallel processing through data splitting requires some preprocessing for data chunking and post process for result aggregation. Those processes result in an overhead not only on execution time but also in developer time, since new code usually has to be written for data chunking and aggregation. Even then, certain processes may not be parallelized due to domain constraints. Therefore, in this stage, execution experts empirically determine workflow bottlenecks in order to identify candidates for parallelization, which, once approved by domain experts, are handled to software engineers for consideration and, if needed, complemented with new components written by code developers.

We have observed in many applications that aggregation steps may be non-trivial to implement, and may require basic research on the domain at hand.

3.7 Create Variants

At this point in the process, when a new workflow template has been validated and tested through several instantiations, we can easily obtain a number of related computations by designing simple variants of the new one. Simple variations include: typical combinations of component parameters and partial instantiations of workflow templates with default datasets.

Of all the possible variants that can be obtained in this way, workflow representation experts must work in collaboration with the scientists to identify those that are meaningful in the domain. While in previous stages different partial models of the workflow elements are built, the modeling goal of this step is to obtain a model for describing the workflow as a whole. This is the model that scientists can use to specify experiments abstracted from the software components used to run them.

3.8 Generalize Workflows

Ideally, in the long term, as iterations proceed a better understanding of the domain can be obtained that allows for a more generative, instead of case-based, workflow design process. For that, workflow representation experts identify commonalities between workflows that can be modeled as abstract components. Those abstractions must be complemented with refinement criteria that let us prefer one specialization over another, given the desired overall workflow characteristics.

This kind of reasoning requires representing connections between the component level models and the workflow level model. Our experience is that this knowledge is easier to acquire when a number of workflow templates have already been designed so that actual examples can be used both to obtain the model and validate it.

4. COST-BENEFIT ANALYSIS OF WORKFLOW APPROACHES

It is important to articulate up front the reasons and benefits of using a workflow system to implement an application. Potential benefits of adopting a workflow-based approach:

- **Provide a clear separation between domain-relevant user concerns and execution details**. This allows for automation and complex optimizations at the execution level, which is especially relevant when the execution environment is a complex one as is the case in grid computing. Furthermore, this layering promotes platform independence and facilitates portability and understandability to domain users.
- **Result validation**. By starting from a well-known workflow template, any analyses are immediately validated because the workflow guarantees the results were obtained using a widely-accepted analysis methodology. The workflow template can be easily reused with different data sets or with parameter variants, facilitating repeatability and reproducibility of experimental results.
- **Accelerate experimental cycle**. Although in the short term adopting a workflow-based approach may require additional effort, once applied it should avoid repetitive work from software engineers, code developers and execution experts.
- **Document experimental results**. The explicit representation of component features, component parameters and data sets for every experimental result is valuable in itself as experiment documentation and as data provenance information for experimental results.
- **Broaden participation in the experimental cycle**. Experienced scientists can focus on developing workflow templates that capture well-accepted methodology, specialized researchers can contribute specific components and models, more junior students can run variants of existing templates and still accomplish interesting discoveries. Without a workflow system, only a few individuals that understand how to manage complex code bases are able to set up and run experiments.
- **Facilitate scaling up**. Scaling up is accomplished through distributed execution, parallel execution of data sets, efficient data management services, and seamless transition to executing with high-end computing resources.

Once it is understood which of these benefits are important to the application, we assess the cost of the workflow-based approach. We have very informal ways to estimate this cost by discussing which work needs to be done at the different stages given where the status of the application. Therefore, understanding the overall process and what will be required at each stage becomes very important to a prospective user of workflow systems.

Sometimes the cost-benefit analysis does not justify the cost of converting an existing application into a workflow. The existing codes and scripts may be flexible enough for what is needed. The investment of converting them to a workflow system is only worth it if there is a clear need to exploit the advantages. By extending current workflow systems to automate and assist in all tasks

during the workflow creation process, this cost would be greatly reduced. This would lower this important barrier for adoption and facilitate widespread use of workflow technologies.

5. RELATED WORK

Computational workflows are software artifacts, so the process for designing them we have just described can be analyzed as a methodology for software construction. From that point of view, our methodology has some commonalities as well as distinct requirements that are worth considering here.

In our experience, a key distinct feature of computational workflows as software artifacts is that they are often designed as its development is taking place because the design space is not necessarily well understood. Multiple models may exist to represent a particular domain problem and/or several algorithms to solve different aspects of the same problem without a clear understanding of their interactions. The approach that we take is to let the user play an active role in the development process, very much in the line of agile methodologies for software development which are especially well-suited for building software from poorly specified changing requirements [20]. Agile methodologies propose close customer collaboration during the whole development process so that requirements can be refined as needed and prototypes promptly validated as they come out. However, we depart from agile methodologies in that instead of having synchronous communication with the end user, who must stay almost full-time with the development team, we intend to provide the users with tools for experimentation by letting them build their own variations of the system (i.e., their own workflows). In order to make this possible, we take a model-based approach where users specify requirements in their own domain terms and build a model of the workflow that is then transformed into more detailed models and finally executed.

The Model Driven Architecture (MDA) [21] as defined by the OMG consortium shares with the approach presented here the interest on separating the problem domain from the execution environment in a software system. MDA proposes the use of different models for a software system along with a number of transformations going from more abstract models into more specific ones. In MDA terms, its goal is to separate business and application logic, which tends to be more stable, from the underlying platform technology, which may evolve more quickly due to technological evolution. MDA is intended mainly for large-scale distributed web-based business applications. The main differences with our approach come from the type of models used to describe a software system. MDA models are built with UML and do not support complex semantic constraints other than is-a and part-of hierarchies. On the other hand, UML models allow expressing not only data flow aspects of an application but also module relations and operation sequencing. Finally, while MDA tries to model the whole software system from a very high level down to the function level, a workflow system considers course-grained component level descriptions of data and execution requirements without concern for how those components are implemented internally.

Workflow-based approaches promote reuse at different levels: code reuse, since algorithms are componentized they can be plugged into different workflows; design reuse, since every workflow template provides an abstract design that can be specialized with different components; requirements reuse, since the formal description of the problem solved by a given workflow documents interesting combinations of problem characteristics; and test reuse, since workflow instantiation and execution are also record for future use. However, as it is well documented in the research on software reuse [22], any methodology that promotes reuse, and somehow prioritizes a long-term view of software development that penalizes short-term results, must take into account not only technical but also human and organizational factors in order to be successfully applied. We advocate what in the software product lines literature is known as "minimally invasive transitions" [23], trying to minimize disruption of ongoing development efforts and to take advantage of existing software assets to make possible an incremental adoption of the workflow-based approach.

6. SUPPORTING WORKFLOW DESIGN

The workflow tools that our group has already developed, Wings and Pegasus, provide effective support for many of these stages of workflow design and execution as has been demonstrated through its use in a variety of scientific applications. Wings is composed of a set of tools designed to support the creation of workflow templates and instances, which are then submitted to Pegasus to create executable workflows. Specialized tools are used to assist users during template creation, while general-purpose knowledge representation tools are used for component model and metadata creation. Wings reasoning mechanisms are responsible for automatically building a workflow instance from a given template and the specification of input data to be used in the computation: a fully expanded workflow instance with the information required by the Pegasus mapping and execution system. Pegasus is a production-level workflow mapping and execution engine. The Pegasus workflow mapping engine maps a given workflow to the resources that are available at execution time, binding data descriptions to one of many possible replicas, selecting hosts to execute the computations, moving data to where computation will occur, and moving data products to data repositories. Pegasus is able to execute workflows in small, medium, and large size grid environments such as the Teragrid and the Open Science Grid.

Our experience using Wings and Pegasus in real applications has lead us to identify a number of bottlenecks in the process that we consider as potential areas of future work on developing additional support for:

- **Defining the role of the workflow system**. Characterizing workflow creation as a programming paradigm would facilitate the process, since it would make clear the level of automation and other capabilities provided by the workflow system. The terms workflow, component, dataset, and code, are often interpreted differently by prospective users. Examples of prototypical workflows, illustrations of bad workflow design or inappropriate uses of the workflow system would also be useful.
- **Initial design**. Migrating into a principled workflow-based approach from existing ad-hoc solutions requires a significant effort from workflow representation and execution experts in order to get an understanding of the domain. Tools for workflow sketching along with light-weight knowledge acquisition tools for rapid acquisition of initial semi-formal models could be useful. In addition, source code analysis tools could be

applied to reengineer existing codes and facilitate this stage of the process.

- **Component modeling**. Building a semantic model of existing software codes is an error-prone activity that could also benefit from supporting tools. Authoring tools should help the modeler by taking care of partially redundant low-level details that now are explicitly represented in general-purpose modeling tools. Additionally, some support is also required for checking the compliance of a given component's code with the model described by the developers of that code.
- **Component versioning**. The evolution of the software used to implement components and the resulting versioning is also an issue that could be supported better. As new versions of components appear that supersede existing ones, workflow templates, component models and execution products related workflow information may require to be updated. Automatic mechanisms to track and manage such dependencies will be important to reduce maintenance costs.
- **Scaling up**. Identifying bottlenecks in execution is now a manual process based on exploring different combinations of datasets and resources for workflow instantiation and execution. The existence of high-level analysis tools to connect execution logs to elements in workflow templates and instances would help in this process.
- **Workflow catalogs**. As workflow-based development for a given domain expands on time and coverage we envision the need for developing mechanisms for managing workflow libraries. More sophisticated mechanisms will be needed for indexing and reusing workflows from a shared library as the number of templates, instantiations and executions increase though contributions from users with different roles in distributed virtual organizations.

Supporting these aspects of the process will be important for making workflow systems more cost-effective and have broader adoption.

7. CONCLUSIONS

Workflow systems that support large-scale computation-intensive scientific applications could revolutionize many sciences. However, their widespread adoption depends on a design methodology that offers enough support and automation to make the process cost-effective. In this paper, we described the process of designing workflow applications and the complexities involved in the various stages of the process that affect the cost of developing workflow applications. We also presented the key benefits of using current workflow technologies for managing computation and scale in complex applications. Finally, we discussed some of the areas where additional tool support would facilitate workflow design. By articulating the benefits of workflow applications and by reducing the cost of developing them, our goal is to make workflow technologies accessible to a broader community of users with applications where computation and scale are important issues.

8. ACKNOWLEDGMENTS

We would like to thank the many collaborators who made various aspects of this work possible. We would also like to thank the National Science Foundation for the support of this work under grants: SCI-0455361, CNS-0509517 and CNS- 0615412.

9. REFERENCES

[1] Taylor, I., Deelman, E., Gannon, D., Shields, M., (Eds). "Workflows for e-Science", Springer Verlag, 2006.

[2] Deelman, E., and Gil, Y. (Eds). "Final Report of the NSF Workshop on Challenges of Scientific Workflows", National Science Foundation, Arlington, VA, May 1-2, 2006. `http://www.isi.edu/nsf-workflows06`.

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

[4] Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger-Frank, E., Jones, M., Lee, E., Tao, J., and Y. Zhao. "Scientific Workflow Management and the Kepler System". *Concurrency and Computation: Practice and Experience*, Special Issue on Workflow in Grid Systems, Volume 18, Issue 10, 2006.

[5] Oinn, T., Greenwood, M., Addis, M., Nedim Alpdemir, M., Ferris, J., Glover, K., Goble, C., Goderis, A., Hull, D., Marvin, D., Li, P., Lord, P., Pocock, M. R., Senger, M., Stevens, R., Wipat A., and C. Wroe. "Taverna: Lessons in creating a workflow environment for the life sciences." *Concurrency and Computation: Practice and Experience*, Special Issue on Workflow in Grid Systems, Volume 18, Issue 10, August 2006.

[6] Moreau L. and B. Ludaescher (Eds). "Special issue on the First Provenance Challenge", *Journal of Computation and Concurrency: Practice and Experience*, To appear.

[7] Deelman, E. and Y. Gil. "Managing Large-Scale Scientific Workflows in Distributed Environments: Experiences and Challenges", Proceedings of the Workshop on Scientific Workflows and Business Workflow Standards in e-Science, The Second IEEE International Conference on e-Science and Grid Computing, Amsterdam, The Netherlands, December 4-6, 2006.

[8] Gil, Y., Ratnakar, V., Deelman, E., Spraragen, M., and J. Kim. "Wings for Pegasus: A Semantic Approach to Creating Very Large Scientific Workflows," Proceedings of the OWL: Experiences and Directions 2006 (OWLED-06), Athens, GA, November 10-11, 2006.

[9] Kim, J., Gil, Y., and V. Ratnakar. "Semantic Metadata Generation for Large Scientific Workflows," Proceedings of the Fifth International Semantic Web Conference (ISWC-06), Athens, GA, November 5-9, 2006.

[10] 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." To appear in Proceedings of the 19th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI), Vancouver, British Columbia, Canada, July 22-26, 2007.

[11] Katz, D. S., Jacob, J. C., Berriman, B. G., Good, J., Laity, A. C., Deelman, E., Kesselman, C., Singh, G., Su, M., Prince, T.

A. "Comparison of Two Methods for Building Astronomical Image Mosaics on a Grid," International Conference Workshops on Parallel Processing, 2005 (ICPP 2005), June14-17, 2005.

[12] Maechling, P., Chalupsky, H., Dougherty, M., Deelman, E., Gil, Y., Gullapalli, S., Gupta, V., Kesselman, C., Kim, J., Mehta, Brian Mendenhall, B., Russ, T. A., Singh, G., Spraragen, M., Staples, G., and K. Vahi. "Simplifying construction of complex workflows for non-expert users of the Southern California Earthquake Center Community Modeling Environment" *SIGMOD Record* 34(3): 24-30, 2005.

[13] Blythe, J., Jain, S., Deelman, E., Gil, Y., Vahi, K., Mandal, A., and K. Kennedy. "Task Scheduling Strategies for Workflow-based Applications in Grids," CCGrid 2005, Cardiff, UK, 2005.

[14] Singh, G., Kesselman, C., and E. Deelman. "Optimizing Grid-Based Workflow Execution", *Journal of Grid Computing*, Volume 3(3-4), December 2005, Pages 201-219.

[15] Singh, G., Deelman, E., Mehta, G., Vahi, K., Su, M., Berriman, B., Good, J., Jacob, J., Katz, D., Lazzarini, A., Blackburn, K., and S. Koranda. "The Pegasus Portal: Web Based Grid Computing," The 20th Annual ACM Symposium on Applied Computing, Santa Fe, New Mexico, March 13 - 17, 2005.

[16] Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Koranda, S., Lazzarini, A., Mehta, G., Papa, M. A., and K. Vahi. "Pegasus and the Pulsar Search: From Metadata to Execution on the Grid," Applications Grid Workshop, PPAM 2003, Czestochowa, Poland 2003.

[17] Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Blackburn, K., Lazzarini, A., Arbree, A., Cavanaugh, R., and S. Koranda. "Mapping Abstract Complex Workflows onto Grid Environments," *Journal of Grid Computing*, Vol.1, no. 1, 2003, pp. 25-39.

[18] Gil, Y. "Workflow Composition", In *Workflows for e-Science*, Taylor, I., Deelman, E., Gannon, D., Shields, M., (Eds), Springer Verlag, 2006.

[19] Kim, J., Deelman, E., Gil, Y., Mehta, G., and V. Ratnakar. "Provenance Trails in Wings/Pegasus", To appear in *Journal of Computation and Concurrency: Practice and Experience*, Special issue on the First Provenance Challenge, L. Moreau and B. Ludaescher (Eds).

[20] Mellor, S. J. "Adapting Agile Approaches to Your Project Needs," *IEEE Software*, vol. 22, no. 3, pp. 17-20, May/June, 2005.

[21] OMG. Model Driven Architecture: http://www.omg.org/mda/. Last visited: February/15/2007.

[22] Frakes W. and K. Kang, "Software Reuse Research: Status and Future," *IEEE Trans. on SW Eng.*, vol.31, no. 7, pp. 529-536, july 2005.

[23] Krueger, C. W., "New methods in software product line practice," *Communications of the ACM*, vol. 49, no. 12 pp. 37-40, December 2006.