ABSTRACT

IYER HARINI RAMACHANDRAN. Automation of Scientific Workflow Construction using Templates and Patterns. (Under the direction of Dr. Mladen Vouk.)

Automation of scientific simulation, experimentation and analysis processes is becoming a necessary part of scientific problem solving. Workflows that implement such processes need to work efficiently with large amounts of data, and they need to automate what could normally be a very time-consuming and error-prone task. Design and development of workflows requires both expertise in the scientific domain as well as an understanding of, and facility with, a workflow implementation language(s) and engine. This work presents a study based on a group of workflows developed at Oak Ridge National Laboratory (ORNL) called monitoring workflows. It discusses an approach to monitoring workflow development based on the use of software engineering pattern and template principles. This work examines the similarities amongst the monitoring workflows, and it crafts templates based on the patterns followed by them. It involves the use of relatively high-level parameterized components that can then be composed into more complex entities. A ”fill in the blank” model for building monitoring workflows using templates and patterns is developed and prototyped. It is shown that complex workflows can be built automatically using templates, and that this does reduce information technology overhead for construction of this type of workflows.

Automation of Scientific Workflow Construction using Templates and Patterns

by
Harini Ramachandran Iyer

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, North Carolina

2010

APPROVED BY:

_____	_____
Dr. Munindar Singh	Dr. Rada Chirkova

_____
Dr. Mladen A. Vouk
(Chair of Advisory Committee)

DEDICATION

To

my Parents, Darshu and Tarani

BIOGRAPHY

Harini Iyer was born in Ahmedabad, India. She finished her schooling in Ahmedabad and received the Bachelor degree in Computer Science from Nirma University, Ahmedabad. She worked for Cognizant Technology Solutions, Pune, for a year as a part of the development group for the launch of eBay.in. She joined North Carolina State University for the Graduate Masters Programme in Computer Science in Fall 2008 and joined the SDM group in Spring 2009. Her hobbies include reading, travelling and painting.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude towards my thesis advisor, Dr. Mladen Vouk for his constant guidance, support and encouragement. I would also like to thank Dr. Rada Chirkova for serving on my thesis committee. I am highly obliged to Dr. Munindar Singh for serving on my thesis committee, for his time, for his invaluable suggestions and for guiding me at various stages of my research.

I would like to use this opportunity and thank my colleagues at the SDM center, Norbert and Ustun, and my colleague at NCSU, Pierre, for their constant help and support throughout my research. Special thanks to Norbert, an excellent person to work with, for being so patient with me and answering all my questions.

This work has been supported in part by the DOE SciDAC grant DE-FC02-01ER2548

TABLE OF CONTENTS

LIST OF FIGURES……..vii

LIST OF TABLES……viii

Chapter 1 - Introduction……1

Chapter 2 - Background……..5

2.1 Definitions……5

2.2 Workflows. ……..6

2.2.1 Architecture of a Business Workflow……7

Chapter 3 - Scientific Workflows……10

3.1 Architecture 10

3.2 Lifecycle of a Scientific Workflow 13

3.3 Workflow Users 16

3.4 Benefits 17

3.5 Types of Scientific Workflows 18

3.6 Business Workflows vs. Scientific Workflows 20

Chapter 4 - Scientific Data Management Center……24

4.1 Scientific Process Automation (SPA) 26

4.1.1 Framework for Integrated End-to-End SDM Technologies and Applications [FIESTA].. 26
4.1.2 Ptolemy II 28
4.1.3 Kepler 29
4.1.3.1 Architecture of Kepler System 30
4.1.3.2 Features of Kepler..... 33

4.1.4 Outreach and growth of SPA 35
4.2 Case Study Scientific Workflows 39

4.2.1 Automation Tasks of the Monitoring Workflows 40
4.2.2 Features of a Monitoring Workflow 41
4.2.3 XGC Simulation Code 42
4.2.4 GEM Simulation Code 43
4.2.5 Pixie3D Simulation Code 43
4.2.6 Analysis of the Monitoring Workflows 43

Chapter 5 - Workflow Templates and Patterns……..45

5.1 Basic Workflow Patterns 46

5.2 CPES Generic Template – ProcessFileRT actor 48

5.3 Our approach to templates 50

5.4 Templates for the Workflow Generator Wizard 51

5.4.1 Preparation Steps 52
5.4.2 Post-processing Steps 53
5.4.3 Performance Monitoring Steps 54
5.4.4 Processing Steps (Auto-generated Template) 55
5.4.5 Miscellaneous Templates/Generic Actors 57

Chapter 6 - Workflow Generator Wizard……..59

6.1 An XML Representation of a Workflow 60

6.2 Architecture 62

6.3 Implementation 63

6.3.1 Web Form 64
6.3.2 Library 66
6.3.3 Generator 66
6.3.4 Performance Analysis and Testing 67
6.3.4 Availability of the code 72

Chapter 7 - Conclusion and Future Work……..73

References……..75

Appendices……..82

Appendix A – XGC **83**

Appendix B – GEM **85**

Appendix C – PIXIE 3D **87**

Appendix D – ProcessFile Actor Parameters **89**

Appendix E – VisIT interface **90**

Appendix F: Acronyms and Defintions **91**

LIST OF FIGURES

Figure 1 – Architecture of a Workflow Management System [WMC95] 8
Figure 2 – System Architecture of SWfMS [Lin09] 11
Figure 3 – Lifecycle of a Scientific Workflow [Tan09] 14
Figure 4 - SDM layered architecture [SDMC09] 25
Figure 5 - The FIESTA framework [Tcho10] 27
Figure 6 - Architecture of the workflow-managed computations [SDMC09] 28
Figure 7 – Architecture of Kepler System [Alti06] 30
Figure 8 – Snapshot of XGC Workflow (Courtesy – ORNL and SDM Center) 31
Figure 9 - Actor in a broader sense 33
Figure 10 – Some basic Workflow Patterns [Bhar08] 47
Figure 11 – Top-level view of the ProcessFile actor [Podh07] 49
Figure 12 – The Performance Processing pipeline of the GEM monitoring workflow. Red boxes indicate the ProcessFile actor 50
Figure 13 – Translation of the Preparation Steps of the Monitoring Workflow into a Template 52
Figure 14 – InitTemplateActor Template 53
Figure 15 – Translation of the text archival pipeline into the TransferArchiveText template 54
Figure 16 – Translation of the image archival & movie making pipeline into the MakeMovieTemplate template 54
Figure 17 – Translation of Performance Processing pipeline into a template 55
Figure 18 – Auto-generated template, Diagnostics_3D for the 3-D pipeline of the Pixie3D workflow 57
Figure 19 – Auto-generated template, Diagnostics_1D for the 1-D pipeline of the XGC workflow 57
Figure 20 – MoML view of the PN Director and the Logger actor 61
Figure 21 –Architecture of Workflow Generator Wizard 63
Figure 22 - The web form table 64
Figure 23 - The actor_process_file table 65
Figure 24 - Auto-generated Pixie3D Workflow 67
Figure 25 - Auto-Generated XGC Workflow 71
Figure 26 - Auto-Generated GEM Workflow 72

LIST OF TABLES

Table 1 – Existing Templates in the Monitoring Workflows ……. 57
Table 2 - Comparison between manual generated and automatically generated XGC Workflow ……..……..……..……. 69

Chapter 1 - Introduction

This thesis is about automation of workflow services, specifically those associated with scientific computing processes that are managed over the network.

Internet or World Wide Web (WWW) [QiXu08] is a complex network of interconnected dynamic and distributed elements consisting of repositories of data, software and services. In recent years WWW has transformed from basically being a referential database to a congregation of interacting applications available as services [Sing94, Sing96, Sing05] – transformation into a "cloud computing" environment is now very visible [e.g., Vouk09]. A web service can be rendered using different technologies such as SOAP [SOAP00], WSDL [WSDL01] and UDDI [UDDI10]. Service Oriented Architecture (SOA) is at the heart of such services [Sing05]. The SOA model is implemented to make the components as reusable as possible. It ensures the compatibility amongst the various web services, giving them a common platform for interaction [IBM10, IBM010]. It provides interfaces, which facilitate communications among applications independent of the programming the languages or tools used to develop these applications.

A workflow is a complex entity. It can use web services, but it does not have to. It is a collection of connected units (processes), which define a sequence of actions for a system. Operations are performed through a set of distributed units. In the context of this thesis, and based on the Kepler [Alti04] environment, these process units are called *actors*. A workflow depicts the "work" as a repeatable sequence of actions that can be translated into script or a graph (including a graph with feedback loops). A workflow modeling system based on SOA has proven to be a powerful combination for applications demanding systematic execution in a heterogeneous environment [Sing94].

The workflow model is extensively used for business processes to make it easy for organizations to understand, execute and monitor the services they use, and the interaction amongst these services. While workflows can be described in many ways, in the business community they often use Business Process Execution Language (BPEL) [OASI07] as a way of defining the business processes and interaction protocols.

As the computing capabilities transformed to become more powerful and the management of the associated data more complex, computational scientists had to face the challenges of managing the processes and data flows produced during simulation and data analysis. A challenge that presented itself was a way to automate the scheduling, execution, analysis and visualization processes involved in such workflows. This need for automating complex scientific computations gave rise to *Scientific Workflows* in the sense this concept is used today.

A *Scientific Workflow* can be defined as an information assisted workflow constructed to schedule and execute scientific simulations and experiments in a distributed way, and of analyzing, processing and visualizing the data produced in the process [Sing94]. Scientific workflows were introduced keeping business process workflows as the base, while dealing with enormous data and computer intensive tasks frequently found in scientific applications [Bark08, Critch10, Vouk07]. While business workflows tend to be relatively slow changing and stable, a characteristic of scientific workflows is that they might change more frequently – and domain scientists often make those changes.

The different scientific workflows used by the Department of Energy (DOE) and studied for this document can be located on the Scientific Discovery through Advanced Computing website [SciDAC10]. They include workflows that manage large-scale astrophysics, fusion, combustion and climate simulations. It is interesting to note that, not unexpectedly, scientific workflows falling under the same scientific discovery domain tend to be similar in what and

how they do their work. For example, they all deal with very large amounts of data produced by a simulation or a scientific experiment, they all move the data from supercomputers to analytics clusters where the data is further analyzed and visualized, etc. Unfortunately, while similar, such workflows also have distinguishing features, which can make the re-use of the workflows difficult [Crit10]. This can result in a considerable amount of rework for the workflow designer. To complicate matters, given today's workflow technologies, creation of new workflows and/or adaptation of existing ones requires not only advanced understanding of the domain science and processes, but also a relatively advanced knowledge of the workflow implementation technology. The latter is an investment many of the scientists are unwilling, perhaps unable, to make. So, in practice, workflow development usually necessitates involvement of computer scientists as workflow designers.

The intent of the work described in this thesis is to discuss potential approaches to automation of the process of creating scientific workflows. The focus is to ease and reduce the information technology overhead on the workflow designer during the task of creating workflows that are similar to the ones that are already in existence. In doing so, this work leverages the software engineering concepts of patterns [Gamm95] and templates [Barr94]. This project includes investigation the different scientific workflows currently used by some DOE computational scientists, identifying patterns followed by them, and defining a template-based approach to implementation of such workflows.

This thesis is divided into the following chapters. Chapter 2 (Background) explains some basic definitions, and discusses the concept and characteristics of Workflows in detail. Chapter 3 (Case Study Scientific Workflows) describes the concept of scientific workflow and discusses the similarities with the Business Process workflows. Chapter 4 (Scientific Data Management Center) discusses in detail different types of scientific workflows that are part of the current case study. Chapter 5 (Workflow Templates and Patterns) talks about the role of templates in workflow development. It also characterizes the different templates

created, and the methods used to create them. Chapter 6 (Workflow Generation Wizard) discusses the proposed workflow generation wizard architecture, and the results of its pilot implementation. Chapter 7 (Conclusion and Future Work) provides a summary and discusses possible future work.

Chapter 2 - Background

2.1 Definitions

The definition of the workflow has evolved with the evolution of workflows. The Workflow Coalition Group [WMC10] defines a workflow as "*The computerized automation of a business process as a part or the whole*". It is a sequence of actions performed through connected entities each representing a specific task. A process is an activity, which involves transition, translation and transfer of data and control, which may or may not involve a set of sub tasks, in order to achieve a predefined goal. Many times, a workflow and a process are considered to be equivalent. However, there is a thin line between process and workflow. A workflow can be considered to be one of the many possible ways of implementing a process (or a group of processes) in order to achieve the goal. As described in [Sing05], *a workflow is a narrower concept than a process*.

A workflow management system is "*A system which defines, manages and executes workflows, through a computer software which has a workflow logic defined to decide the order of execution*" [WMC10]. A workflow engine is a software development tool that is used to model, develop, execute and monitor different workflows. There are many workflow engines that exist today, for example, Kepler [Alti04], Triana [Tayl07], Windows Workflow Foundation [WWF01] and Pegasus (DAGMan) [Deel05, Deel08]. The concept of workflow technology emerged as a result of the growing need for avoiding the repetitive (and often error prone) sequence of actions in business processes thereby making the workflow technology a welcome solution for the business world. With the enormous expansion in the scientific world, the experiments have become more detailed and more complex, and hence their simulations produce the data in larger and larger amounts – nowadays in terabytes and petabytes. This led to the emergence of a new dimension of science called eScience.

eScience is "*a computationally intensive science that works with enormous data sets in a distributed network environments*". The different applications belonging to the same domain of eScience were studied [Bowe06] and it was observed that they follow the same or very similar protocols. This realization led to the need for workflows to be extended to the field of science giving rise to a new dimension of workflows called Scientific Workflows. A scientific workflow is "*a workflow designed exclusively for the eScience applications to execute a series of computational, data manipulation and efficient data storage tasks*".

2.2 Workflows

As already mentioned, the concept of *Workflow* and *Workflow Management Systems (WMS)* was realized with a motive of automating simple office activities like document management. In late seventies and early eighties, before the computational systems were introduced in business organizations, any services offered by the organization would lead to a lot of paperwork, which added to the complexity of the system, giving rise to a highly complicated data and control flow [Stoi06]. The reason behind this was the lack of co-ordination between the different tasks, each of which was treated as an independent task.

For example, as discussed in detail in [Brah07], a customer at Danske bank had to fill out and sign a different form for each account the person needed to open. This led to a lot of paperwork for the customer, which was equally unmanageable for the bank. In order to eliminate this overhead on both the sides, they used software designed by IBM [BPEL07], which was an automation of the entire process. This reduced the complexity on the customer's side, giving the customer a better interface with the system and relieved the management from the tedium of processing a lot of paperwork. This lead to the idea of integrating computational techniques with the business processes in order to improve and optimize the resource consumption by automating the business process. The concept of workflow, since its establishment, has for the most part been associated with business

processes and business process re-engineering. The intent of using the workflow technology to implement business processes was to keep the process level and the application level independent of each other. Implementing business processes, irrespective of their size, through workflows was a welcome solution to all, which led the Workflow Management Coalition to development of a reference model [WMC95] in order to accommodate different workflow implementation techniques. This body proposed an abstract model with the techniques to mold the processes into a workflow model. We now discuss this architecture.

2.2.1 Architecture of a Business Workflow

The following are the characteristics of a workflow specified in the WMC Reference Model document [WMC95] and the associated relationships.

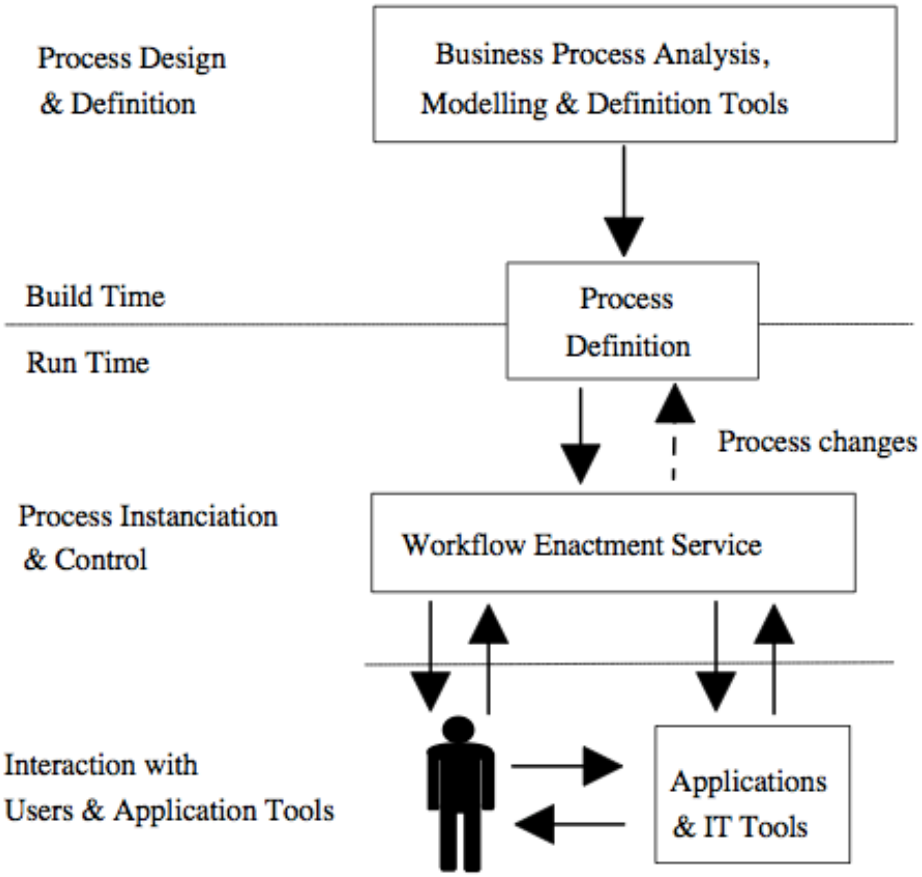


Figure 1 – Architecture of a Workflow Management System [WMC95]

A workflow model is divided into two main types of functions and an interaction layer,

- Build-time functions
- Run-time functions
- Interaction with Users & Application

The build-time functions include defining and modeling the business process in terms of a workflow model and its activities. Once the process is defined, it triggers the runtime functions. These interpret the definitions and start building the operational workflow instances, schedule the activities as desired, and controlling them. The interaction layer is responsible for the input by the users through the various applications available, in order to

process the activities. Experience has demonstrated the effectiveness of using Workflow Management Systems (WMS) for modeling purposes in the business process management community [WMC95].

In this context, it is worth noting that Kepler [Alti06, Kepler04] is an example of a WMS for scientific workflows. Kepler-based processes are called actors and the primary control is data-flow based. Each actor is responsible for a particular task. Kepler and actors are discussed in more detail later in the next two chapters. In general, a complex workflow can be decomposed into a collection of partial workflows [Vouk07]. Conversely, a complex workflow can be built by the combination of one or more simpler workflows. The following chapter discusses scientific workflows in detail.

Chapter 3 - Scientific Workflows

There are different ways of defining scientific workflows. One is to directly adapt the definition given by the WMC described in the previous section. Another one derives from [Alti08]: Scientific Workflow is "*A representation of technology that is built to automate scientific process(es) with the goal of reducing information technology overhead and facilitate use of the technology* ". As already mentioned, the idea behind scientific workflows is to hide lower level information technology details and complexities, such as repeated file movement actions, from scientists and give them a better support to carry on their scientific work. This chapter discusses architecture, benefits and lifecycle of a typical scientific workflow encountered in a certain type of Department of Energy scientific activities, and compares it with a typical business process workflow.

3.1 Architecture

Automating scientific experiments and simulations by enacting them into scientific workflows is challenging and can be error prone [Alti03]. There is a set of requirements scientific workflow support technology needs to meet in order to be usable in various science applications. These requirements can be summarized as follows [Luda06] –

1. Seamless access to information technology resources and services
2. Ability to provide service composition, reuse and workflow design
3. Scalability
4. Detached execution
5. Reliability and fault-tolerance
6. User interaction (ability to allow user to interact with the workflow in a friendly way)
7. Smart reruns
8. Provenance and meta-data collection

As discussed in [Lin09], the architecture of a Scientific Workflow Management System (SWfMS) can be divided into the four layers shown in Figure 2.

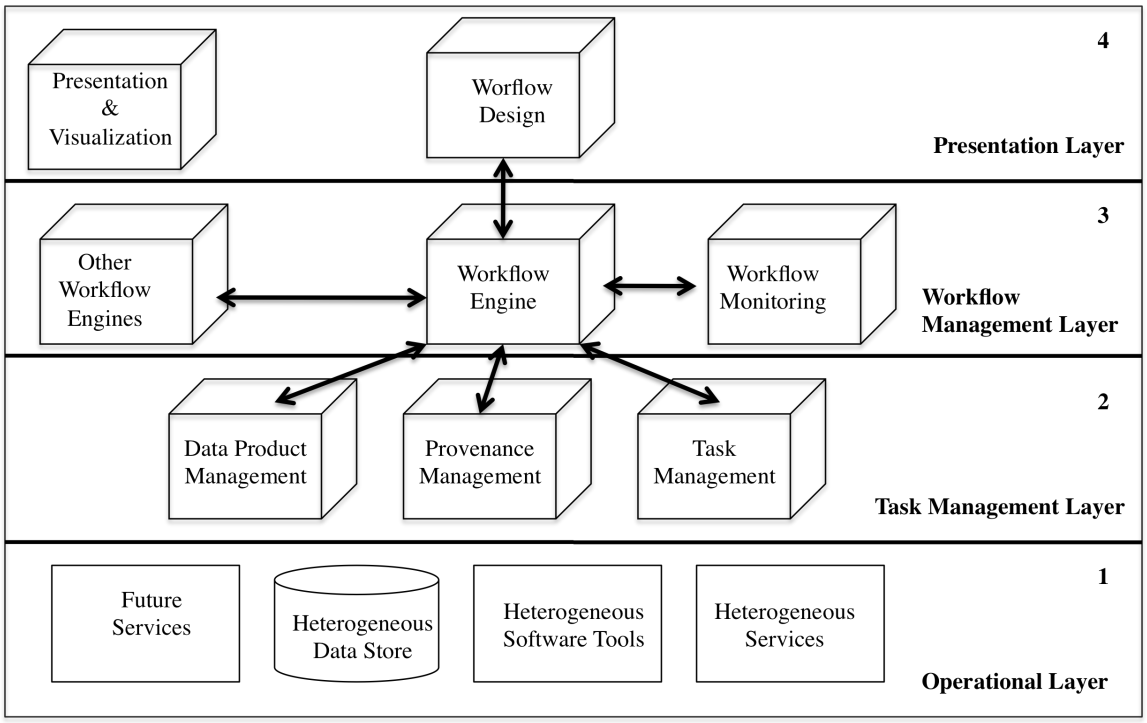


Figure 2 – System Architecture of SWfMS [Lin09]

Layer 1 – Operational Layer

This layer consists of a variety of heterogeneous basic software, hardware and storage services and tools. These resources are integrated via the workflow system in order to facilitate access to them, and enable the computations and the data management a scientist needs to perform. Scientist can choose among different available services to satisfy the scientific processes that need to be executed.

Layer 2 – Task Management Layer

This layer is responsible for making the execution specific to a process. It provides basic process elements integrated in Layer 3 into an actual workflow. These tasks are the building

blocks of a scientific workflow system. Each task is responsible for taking the input tokens given to it, and producing relevant output tokens. Task Management layer is a strong feature that satisfies most of the requirements of the scientific workflow systems [Luda06]. The data product management module in this layer is responsible for the efficient management of the data and the provenance metadata. The provenance management records information like the history of the processes, the output tokens produced for the input tokens, the environment in which the simulation is executed and other such information that imparts the reproducibility feature to the system. It ensures data provenance by recording the data and the system configuration involved in the workflow *run* by generating reports in, for example, XML or HTML. The provenance support enables a variety of higher-level functionalities such as fault-tolerance [Moua10, Moua09]. An example is the '*smart re-run*' feature [Craw08] that uses the data provenance and re-executes the affected part of the workflow in case of an abnormal system state. In addition to enabling fault tolerant and auditability, provenance information can facilitate a number of other higher-level functions such as optimal resource utilization.

Layer 3 – Workflow Management Layer

This layer is responsible for workflow data and control flow management, i.e., the execution and monitoring of the scientific workflows. It co-ordinates the tasks provided by the Task Management Layer and executes the scientific workflow. Each of these workflow execution cycles with a set of tasks is called a *task run*. In the context of the Kepler workflow systems these tasks are called "actors" [Alti04]. When the scientific workflows are long running, it is imperative to record and visualize the progress of the workflow evolution [Barr07]. The data monitoring involves tracking the data produced by the workflow and perhaps using the visualization tools such as *AVS* [AVS10] or *VisIT* [Viz10] to plot the images. The isolation of this layer from Layer 2 ensures the standardization of the workflow models making them reusable and interoperable between other workflows and sub workflows. The abstraction provided by this layer separates workflow scheduling from

individual task execution thus making the workflow system more scalable and thereby improving the performance.

Layer 4 – The Presentation Layer

This layer, as its name implies, facilitates user access to workflows and enables customization of the user interfaces and visualization of the workflows. The Presentation Layer abstraction allows user customizion of some or all the components of the workflow to suit the scientific workflows in the same domain, making the non-customized components re-usable. Thus, we can say that this layer helps in standardizing the workflow layout and enables the user to customize the workflow.

To summarize, Operational Layer provides seamless access to resources, services, technologies and tools, which a scientist can choose from. Layer 2 instantiates tasks and provided building blocks that are then integrated into a workflow in Layer3. The task management layer addresses the requirements 5, 7 and 8 by enabling reliability and reproducibility through data provenance, smart re-run features and so on. Layer3, the workflow management layer, addresses requirement 4. Layer 4 addresses the requirements 2, 3 and 6. The presentation layer helps to design workflows in a more abstract way, standardizing most of the functionalities, making the system reusable and scalable. The detachment of the workflow management layer from other layers makes the execution independent of the task model.

3.2 Lifecycle of a Scientific Workflow

The workflow lifecycle is about definition, development, evolution, deployment and execution of scientific workflows [Tan09, Shos09]. The following figure depicts a typical scientific workflow lifecycle.

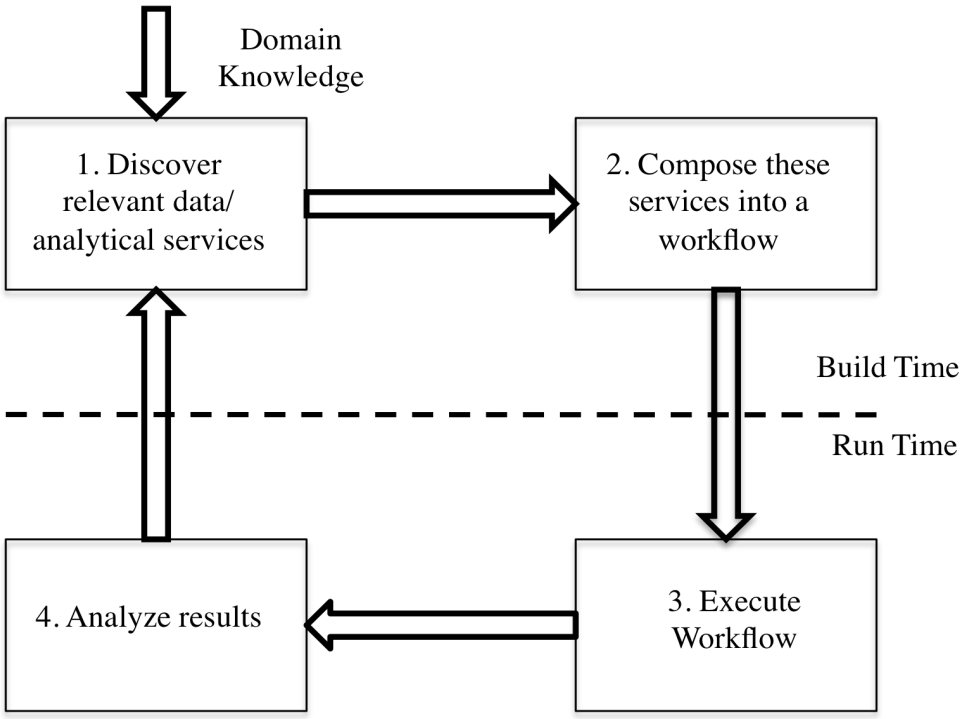


Figure 3 – Lifecycle of a Scientific Workflow [Tan09]

The workflow lifecycle is divided into two sets of functions, build-time and run-time functions, which in turn span four phases of a scientific workflow life. Build-time functions include activities such as defining a scientific application and the requirements for enacting it into a scientific workflow, collecting the information regarding the data and services that are required, and so on. Once the data and the processes and analytic services are identified and defined, the details are translated into a workflow. The run-time functions include executing the scientific workflow that has been constructed and analyzing the results thereby produced. The four main phases that a scientific workflow lifecycle normally consists of are;

Phase 1 – Service Discovery and Resource Planning

The workflow construction starts with the data analytics and requirements gathering from the scientists and building of a detailed specification list based on the detailed requirements.

While scientific communities tend to be autonomous, and have their own set of science related processes, information technologies and services are more generic but open to the customization according to the scientific problem under consideration. This phase involves domain scientists and workflow designers who identify the technology and the services that are needed to enact the scientific procedure (e.g., simulation of plasma fusion processes) into a workflow. The domain knowledge pertaining to the syntax and semantics of the technology services are acquired during this stage to then be mapped onto information technology components and data and control flows.

Phase 2 – Workflow Design and Composition

Once the requirement specification is acquired and the services have been identified, the workflow designer starts building the workflow. A workflow is composed of components, which are entities that could be atomic, or could encapsulate one or more entities within themselves. Such a composite entity could be a workflow in itself that is responsible for a particular task and acts as a re-usable component, or sub-workflow, for other workflows. Typically, a scientist, or a workflow designer, starts constructing a new workflow by either assembling the relevant components or sub workflows from the repository, or by modifying the existing workflows to make it work for the requirements in hand. Once the workflow has been constructed, and certainly prior to the execution, resource planning functions are carried out. These include workflow validation, data binding, parameter binding, scheduling and optimization. In context of the scientific workflows developed by the Department of Energy (DOE) Scientific Process Automation (SPA) team[1], the workflow design phase may also include scheduling high performance computing (HPC) resources such as supercomputing, grid or cloud computing resources prior to the workflow execution, as well as data distribution management across the network.

[1] The author of this thesis has been a member of SPA over the last two years. See next chapter for more detail,

Phase 3 – Workflow Execution

Once the workflow is designed and composed (i.e., workflow components are suitably interconnected via communication channels), the parameters of the abstract workflow that is being customized for a specific scientific application are mapped onto the parameters of the task model for the execution. Workflow execution is triggered by a signal, which could be a scriptted in a language like Python, or in some other way. During the execution, the data is consumed as data tokens, and the output is generated as data tokens. Depending on the type of workflow, provenance and other meta-data information is generated and collected in a provenance store. The data dependencies that are recorded along with the environmental variables help manage and visualize the dynamic evolution of the workflow. They also help in formulating and managing reliability, recovery, security and other services.

Phase 4 – Workflow Execution Analysis and Result Sharing

Execution of workflows yields both execution results and information (meta-data) about the execution itself and about the transformation effected in the input data. Post execution analysis of the workflows themselves, as well as of the generated outputs, of course is very important. Scientists using workflows often need to inspect and analyze the workflow runs in terms of the time taken by the workflow to execute, debugging in case of failure or abnormal termination, and performance analysis of the workflow for improvements. On successful and satisfying execution, scientific workflows can be shared with other groups and scientific communities by storing the workflows in the repositories, making them accessible to all, and/or by sharing the results of their execution.

3.3 Workflow Users

In the context of workflow management we distinguish four levels of users: 1) **domain scientists** who just use existing workflows by modifying their input data and parameters, 2) more **advanced users** who may modify existing workflows and compose simpler new

workflows using an interface, a repository of workflow components, and perhaps workflow templates and patterns, 3) **workflow designers** who develop new workflow components, templates and patterns, and construct/compose complex workflows, and 4) **workflow system developers** who develop workflow execution engines and workflow support functions (e.g., fault-tolerance).

Level 1 users: *Workflow Operators* – they are the vast majority of end-users. They are scientists whose focus is domain science and the related applications. They require the workflows to conduct their scientific experiments and often seek workflow engineers (Level 2 or 3 users) to help them build the workflows. They submit, execute, monitor and analyze their experiments by operating the workflows by filling in the required parameters, but in general, they are neither interested in the underlying technology nor in construction of workflow elements or technologies, just in their use to further their domain science.

Level 2 and 3 users: *Workflow Designers* are often scientists, or groups of domain and computer scientist, who understand both the requirements of the domain and of the workflow and use their domain knowledge to design and develop the complex workflows fulfilling the requirements. They use the workflow engines like Kepler [Alti04, Kepler04] to build the workflows. The work described in this thesis is primarily concerned with support that would facilitate the work of level 2 and 3 of users.

3.4 Benefits

The higher-level functionalities and the extensibility provided by the scientific workflow systems – and the subsequent reduction in information technology overhead, has lead to a considerable increase in acceptance of workflows by Level 1 and 2 users. Some of the advantages listed in [Shos09] include:

- **Automation –** A scientific workflow is an easy way for automating repetitive tasks. A scientific experiment has to be carried out with different sets of parameters

numerous times. It is tedious to manually launch and monitor every execution run of the experiment - inputs, outputs, all intermediate steps, etc.. Enactment of the experiment into a scientific workflow eases the scientist of the laborious task, letting him concentrate on the science aspect of the experiment rather than the computational side of it.

- **Provenance and Reproducibility –** Provenance information is a powerful tool that enables run-time features such as fault-tolerance, post-run analyses, and measurable reproducibility of rounds. Automatic collection of that information through a scientific workflow system makes such systems convenient and popular.
- **Monitoring –** Run-time monitoring and visualization of the workflow execution and of the results of the execution helps in steering computations and interpreting workflow performance.
- **Optimization and Efficiency –** A smart design of a scientific workflow, especially a complex one, can lead to considerably more efficient use of computational resources, of the scientists' time, as well as energy savings. For example, many scientific workflows are highly distributive in nature, and using their '*parallelism*' nature effectively can improve the performance of the application.
- **Reusability –** Workflow artifacts, such as component entities (or actors in the Keper context) and sub-workflows can be shared across various scientific domains and groups through a commonly accessed repository. This helps the scientists to use the existing components instead of building the same feature from the scratch. This translates into direct support for Level 2 and 3 users.

3.5 Types of Scientific Workflows

Scientific workflows may belong to more than one category. In this section we focus on the types of scientific workflows developed and used at the SDM Center [Chapter 4]. Based on factors like the frequency of execution, the size of input datasets and the amount of data

produced, it may or may not be advisable for scientific applications to be translated into a single workflow. SDM Workflows are distinguished into two categories: a) **simulation** workflows that prepare and launch execution of experiments/simulations, and b) **monitoring** workflows, used to provide run-time monitoring and possibly for the post processing functions of the datasets produced by the simulation workflows. Also, we can distinguish data intensive or compute intensive workflows. Other classifications are possible.

For example, classification can be done on the basis of whether they are individual process (actor) intensive or visualization intensive [Bowe05]. Another classification, been proposed in [Shos09], is based on what the components of the workflow represent or model. This makes the workflows either service oriented or resource oriented.

Simulation Workflows - The DOE is involved in building scientific workflows for its applications, which need to be executed frequently and/or on a regular using different parameters and datasets. Such workflows are considered to be ***simulation workflows*** that need to be automatically prepared and then execution needs to be started – usually on supercomputers. Such workflows are often developed specifically for computation-intensive science experiments like environmental monitoring and fusion simulation experiments. Such workflows may also have a monitoring component, i.e., while they run their state and outputs are monitored to gain insight into the progress of the computations and potential steering of the computations. Outputs and provenance information may be automatically stored and archived, and the data moved to analytics and visualization clusters for further processing. However, in some cases such workflows are started manually, and only the monitoring part is turned into a workflow.

Monitoring Workflows – In some situations, in order to make the task of data management easier, the data produced by production workflows is fed to a different workflow called the ***monitoring workflow.*** Monitoring workflows focus on tracking the state of the execution of

computations started outside the workflow, and dealing with the data-intensive post processing functions of the datasets such as copying files, transferring files, archiving images & making movies, archiving data files, etc.

Service (or Domain) Oriented Workflows – Workflows that are developed for the scientific communities to automate the process in an elegant way. The key motive behind these workflows is to enact the scientific process in a way that is comprehensible to the domain scientists. Such workflows are called ***service-oriented workflows*** [Shos09]**.** In such workflows every step is a meaningful domain operation to the scientists. Such workflows are focused on rendering the services rather than the components or resources used for the service.

Resource Oriented Workflows – In contrast, ***resource-oriented workflows*** are focused on the components and the task model used in the workflow rather than the science. The scientific aspects of the application may be hidden from such workflows while the key idea is to optimize the automation part of the process.

3.6 Business Workflows vs. Scientific Workflows

A natural question is how, if at all, are scientific workflows different from business workflows? The workflow idea, started in the business community as a means of automating document tracking and processing [Sing96, Huhn94]. Since, they have been accepted in scientific computational world as well. Business and science domains have different set of applications but many characteristics and requirements are very similar [Bowe05]. However, there are also some important differences.

One of the most important differences is the flow driven approaches that they use. Business workflows tend to be control flow driven, i.e. the business process workflows are

an encapsulation of the different services provided by the business model with decision-making capabilities. Visualization depiction of a business workflow tends to look like a control flowchart. The sequence of processes, usually interconnected with relatively small data streams, is the primary focus. Scientific workflows, on the other hand, are primarily data flow oriented (at least the ones considered in this thesis). Data flows (often quite heavy) dictate the order of processing and data transformations, Workflow management system concept was introduced into the scientific domain in order to make the experiment execution and the result collection easier for the scientists. It deals with the different stages of data processing in a data flow oriented problem solving environments [Vouk97].

A good comparative assessment of business and scientific workflows is given in [Luda09].

Implementation *vs.* Modeling – Business workflows are built as a framework model by analyzing the business requirements. Once the system has been modeled, the service process is implemented partially or as a complete service thereby automating the whole process. The framework serves as a template or a blueprint for the other services to be built upon. On the other hand, scientific workflows are more specific to the requirements in hand. A scientific workflow is generally built as an implementation of a **particular** scientific process that needs to be executed. In many situations, individual business workflows tend to be specific implementations of more general business workflows. In scientific world, at least for now, most of the workflows have not been derived from a generalization, but developed specifically for the case in hand.

It is worth noting at this point that one of the key contributions of the current work is to show that development of a more general scientific workflows, that can then be adapted to specific situations, is feasible.

Experimental *vs.* Business Driven Goals – Scientific workflows deal with automating the scientific processes amounting to massive data. It is purely based on a specific process, collecting and monitoring data at different stages of the experiment. Business workflows are executed with an aim of coordinating the various business and technical levels of an organization contributing towards the goals of the organization.

Users and Roles – Business workflows require more human intervention than scientific workflows. Business workflows might demand distribution of resources to be done manually or they may require a manual trigger after task completion or during the execution of the workflow. Scientific workflows, on the other hand, are built with the intent of minimizing the human intervention and to ease the scientists of the computational tasks making these workflows automated at a higher level.

Dataflow Computations *vs.* Service Invocations – Scientific workflows tend to be dataflow oriented. The flow of data follows the input-output system with data given as input producing an output. Therefore, we can say that the scientific workflows have a *pipeline* structure that typically depicts the flow and transformation of the data. Business workflows, because of their control-flow driven nature, mainly follow a tree structure, with an input that produces an output based on the decision made and appropriate branch of execution is chosen.

Multiple Workflow Instances – Business processes are generally large-scale processes and implementing them into a workflow might require multiple workflows being executed in parallel and combining their results later. Scientific workflows are more linear that way. Multiple instances are not very common in scientific workflows.

We can say that scientific workflows are very similar to the business process workflows in a lot of ways, especially they serve the same purpose of automating the processes in order to avoid the overhead created through re-work. However, scientific workflows do face

certain challenges, which is generally not the case with the business process workflows. Scientific workflows often operate on large-scale heterogeneous data, producing, archiving and processing complex data.

Chapter 4 - Scientific Data Management Center

The US Department of Energy (DOE) launched SciDAC program in 2001 [SciDAC10]. With the emergence of terascale and petascale computing, SciDAC established many research centers and groups to measure and enhance the scientific capabilities that the advanced terascale and petascale computing brought with them to different domains of interest to DOE. Research has been carried on the scientific applications that require or produce such large-scale data. The SciDAC projects involve physicists, mathematicians, computer and computational scientists who collaborate together in the various software application developments, in order to solve the different problems in the area of science [SciDAC08]. Domains which utilize the numerous applications developed include Basic Energy Sciences, High Energy Physics, Nuclear Physics, Advanced Scientific Computing Research, Fusion Energy Sciences, and Biological and Environmental Research [USDOE10]. SciDAC focuses on scientific computing hardware infrastructure, scientific computing software infrastructure, scientific challenging codes, computing systems and mathematical software and collaboratory software infrastructure. One of the SciDAC established centers is the Scientific Data Management Center (SDM) for Enabling Technologies.

SDM center activities [SDMC09] are categorized into the following sets of activities illustrated in Figure 4, borrowed from the SDM center's website [SDMC09], as different layers

- Scientific Process Automation (SPA) – Topmost layer ("glue")
- Data Mining and Analysis (DMA) – Middle layer ("analytics")
- Storage Efficient Access (SEA) layer – Bottommost layer ("infrastructure")

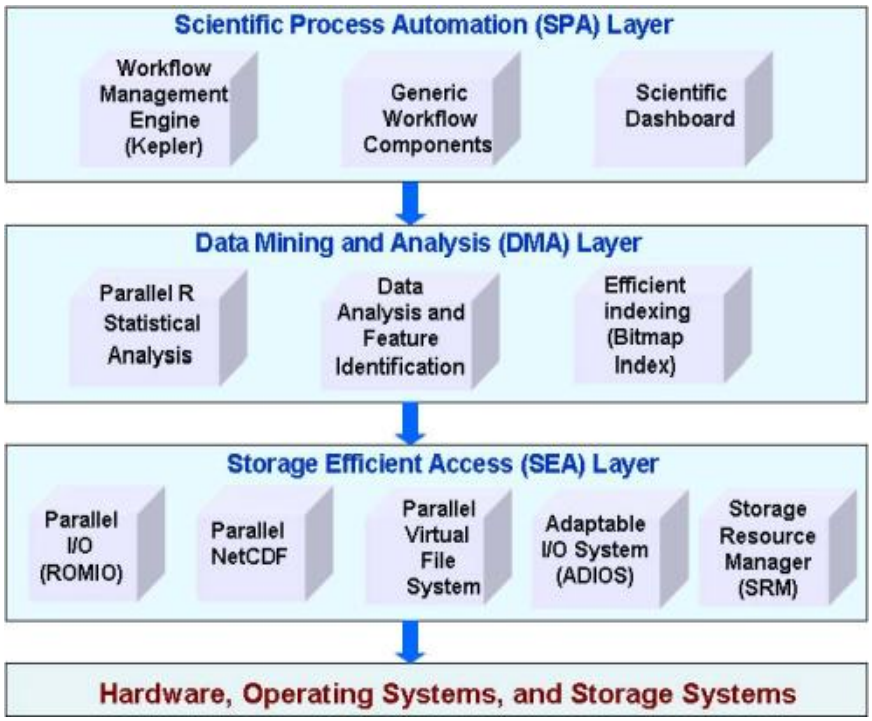


Figure 4 - SDM layered architecture [SDMC09]

One of the major concerns of the scientific computing world is storage and access to the massive amounts of data produced as a result of the scientific experiments. Although the cost of storage has been decreasing, there has been a substantial increase in the computational costs that have to be met as a result of storing the massive data sets. SDM Center deals with this through the SEA layer, which is responsible for the efficient access, reading and writing the huge volumes of data at the minimum computational costs and without maximum benefit to running simulations or other activities, like visualization. SEA performs the functions in a parallel environment.

The DMA layer is all about analytics. Its activities are responsible for providing efficient and effective tools for complex analyses of data. They facilitate efficient algorithms in order

to effectively perform proficient search over the large data sets through specialized feature discovery, parallel statistical analysis and efficient indexing.

The SPA layer, which will be discussed in detail in the following section, is responsible for generating the data by running simulations, storing and visualizing the data. Most of the time SPA workflows are concerned with the post processing of the data that is produced as a result of a simulation. This layer is closer to the user interface and uses different software tools for different activities.

4.1 Scientific Process Automation (SPA)

The SPA project was introduced with the intention of reducing the overhead of the scientists performing computational and data management activities, so that they can concentrate more on the science and the scientific aspect of the problem. Over the years, SPA has developed a number of tools and technologies. Three most successful ones are Kepler, FIESTA and eSimon Dashboard.

4.1.1 Framework for Integrated End-to-End SDM Technologies and Applications [FIESTA]

Department of Energy (DOE) Scientific Data Management (SDM) Center has developed a number of workflows that help DOE scientists conduct their research in the domains of fusion, astrophysics, combustion, etc. Those workflows are implemented in FIESTA (a *Framework* for Integrated. End-to-end *SDM* Technologies and Applications). This framework helps the scientists to run, debug, monitor and analyze the simulations independent of their physical location. Primary interface tends to be a web browser, although some specialized interfaces (such as Ptolemy's Vergil interface) are also available. FIESTA

consists of a number of components that include a fast adaptable I/O, workflow management, dashboard, provenance and metadata capture, visualization and analytics, and data movement components [Tcho10].

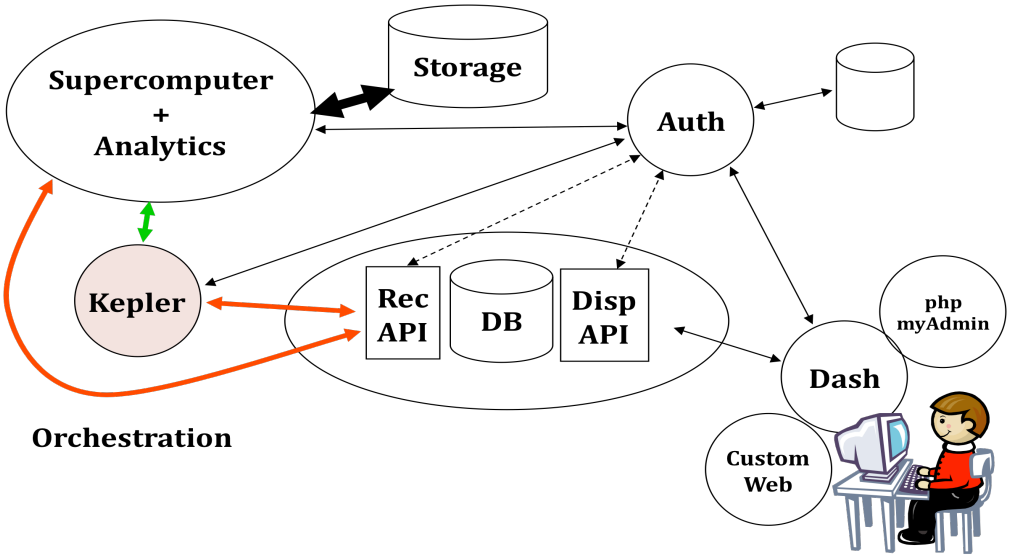


Figure 5 - The FIESTA framework [Tcho10]

FIESTA is illustrated in Figure 5. User submits a simulation job through the web page, or some other graphical user interface (GUI). The workflow engine, e.g. Kepler, Triana, Taverna, etc. then directs the tasks like simulation progress monitoring, output data movement, data archiving, etc..

Workflow relationship to the underlying resources and data collection facilities is illustrated in Figure 6. Control and data flow decisions are made at the workflow control plane, an engine that understands the workflow component relationships and, for example, either sequential or parallel graph-traversal of the workflow data-flows. The actual computations and other resource intensive activities happen at the heavy-duty operational layer. User interface, meta-data tracking, etc. middleware ensures that the users retain insight into the process and have enough information for automatic or manual remote management of both the resources and of the workflow flows.

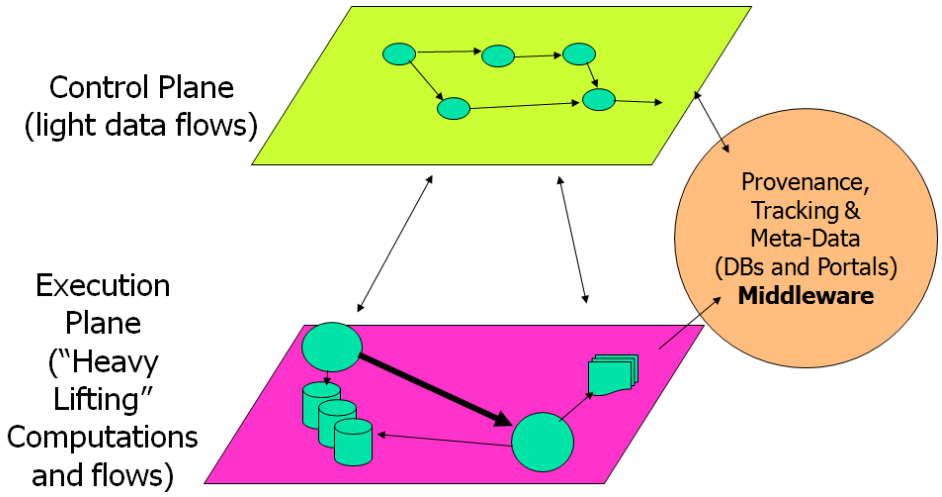


Figure 6 - Architecture of the workflow-managed computations [SDMC09]

The workflow engine that we use in SPA is Kepler [Alti04]. Its architecture is discussed further below. Kepler, an open source project, is based on the Ptolomey open source code.

4.1.2 Ptolemy II

The Ptolemy II [Ptolemy03, Davi99] is the third generation of design software introduced by the Ptolemy team in 1996. It is the successor of the projects Gabriel (1986-1991) [Lee89] and Ptolemy Classic (1990-1997) [Buck02]. Ptolemy II was introduced for heterogeneous modeling, simulation and design of concurrent systems. A model of computation in computational term is the set of allowable operations and costs for a particular computation. The main goal of Ptolemy II[Ptolemy10] is to support the creation and interoperability of executable models under wide variety of models of computations. Ptolemy II has mainly focused on models that are most useful for embedded systems. Most of the models of computations in Ptolemy II support the actor-oriented design, in which the model can be translated into a hierarchical graphs with nodes represented by entities called "*Actors*" and edges representing the dependency between the entities. Every actor has

functionality of its own, and on the successful conclusion, follows the communication channel (the dependency) in order to proceed to the next task to be performed. Ptolemy II offers a unified infrastructure for the implementation of many models of computation. The architecture consists of a set of java packages that provide a generic support for various models of computations. Some examples of these packages are math libraries, graph algorithms, signal plotters and so on. Ptolemy's graphical user interface (GUI) is provided by Virgil. This GUI was adopted by Kepler. It is illustrated in Figure 8. Green icons are processes or actors. Connecting them are data flow paths that transmit tokens, and the black icon with a "horn" is a director – process execution controller. Controllers exist for different execution models, from serial, to concurrent, to continuous time, etc.

4.1.3 Kepler

The Kepler Project, [Kepler04, Alti04], was developed in 2002 by a group of application-oriented research teams with a goal to "*develop an open source scientific workflow system that allows scientists to design scientific workflows and execute them efficiently either locally or through emerging grid based approaches to distributed computation"* [Kepler02]. Kepler is a software application, built upon the Ptolemy II [Ptolemy03] framework, for analysis and modeling of scientific data through workflows. Kepler provides an intuitive GUI (Graphical User Interface) and an actor oriented modeling paradigm [Bowe05], which makes it a powerful and extensible tool for designing, executing and analyzing complex scientific workflows. As already mentioned, the workflows built in Kepler are stored as XML files through Ptolemy's MoML (Modeling Markup Language) [Lee00]. It is a Java based system which has APIs defined to build models from existing Java components called '*Actors*' and the execution of the model thus built are monitored by a component called '*Director'* [Alti06].

4.1.3.1 Architecture of Kepler System

Actors are the entities that encapsulate an entire function or a process. Actors are the building blocks of a Kepler workflow. Most of the scientific workflows concerned with the SDM center are built in Kepler and they typically follow the pipeline structure in which the actors are connected to one another forming a pipe. These actors forming the pipeline communicate with one another by sending and receiving a data product called *token*. The actors communicate via input and output ports, which can be single or multiple. A *Director* on the other hand, determines the model of computation to be used in the workflow. For example, a PN (Process Network) director would mean that the workflow has more than one pipeline which has to be executed in parallel, consuming input and producing outputs asynchronously while an SDF (Synchronous Data Flow) director means that the execution of the pipelines in the workflow is linear in nature because the order of the execution is pre-defined and synchronous in nature. However, looping can be implemented in Kepler, and is frequently used. Figure 8 is a snapshot of the XGC workflow in the Kepler system.

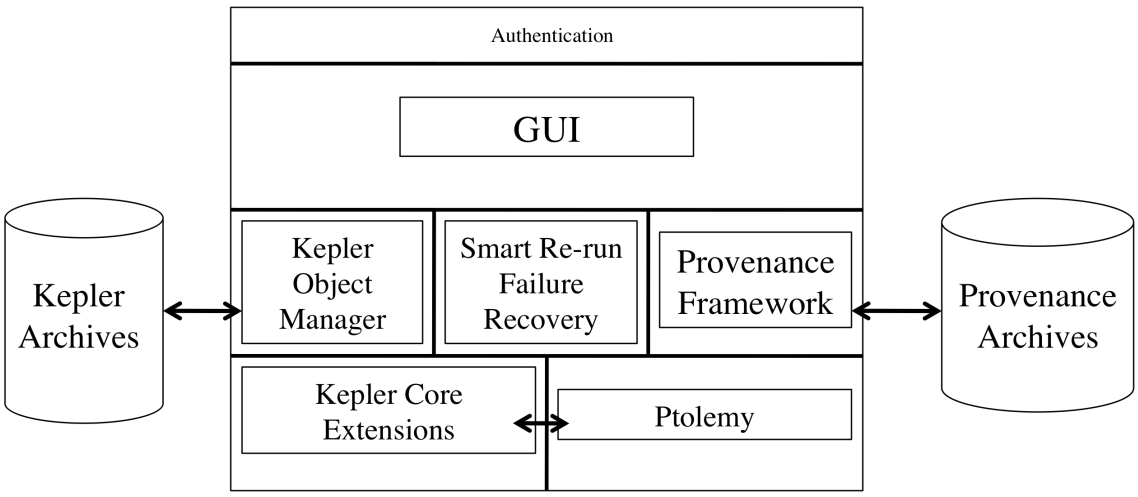


Figure 7 – Architecture of Kepler System [Alti06]

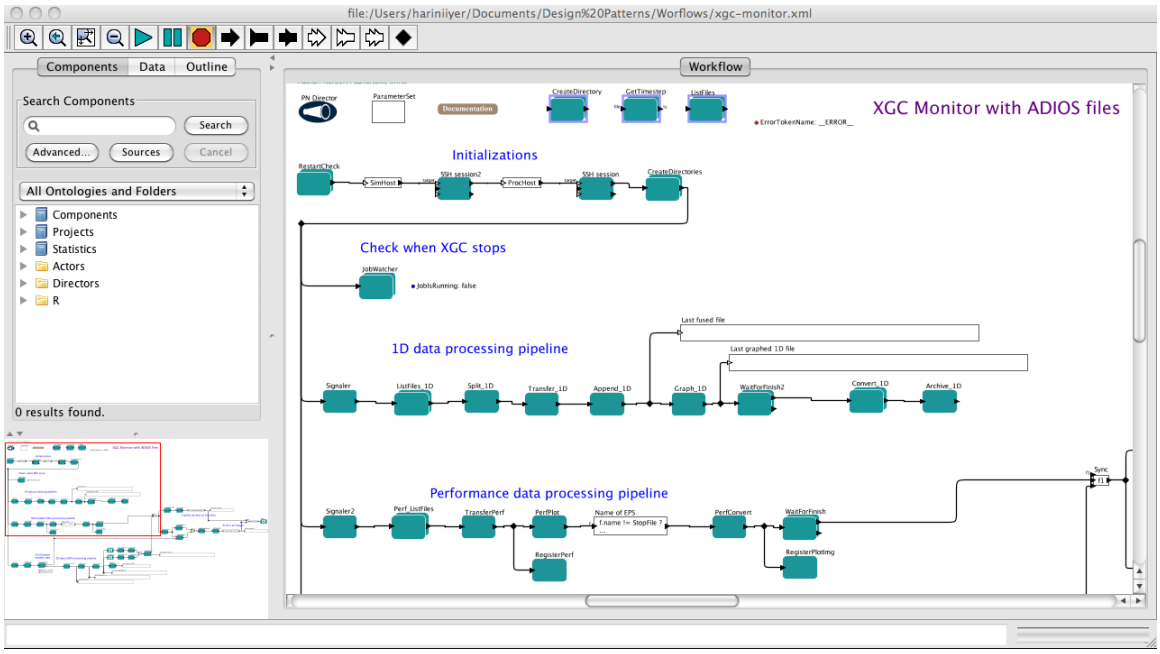


Figure 8 – Snapshot of XGC Workflow (Courtesy – ORNL and SDM Center)

As shown in Figure 7, Kepler system is divided into three levels [Alti06] –

Graphical User Interface – Kepler's GUI acts as a powerful and intuitive tool to create large-scale complex workflows. It has the easy drag and drop feature that makes workflow construction, an easy task, if the design is known. However, developing the workflow takes searching for the right actor in the Kepler library, dragging and dropping the actor on the workflow canvas and knowing how to connecting the relevant ports. The interface is convenient and supportive for grouping the actors into a single entity called '*composite actor*', making it multi-layered and easy to understand.

Middle layer – Kepler separates the components for building the workflow from the orchestration of the workflow, which makes it easier for the workflow designer to visualize the development of the workflow and decide the relation between the components. The middle layer maintains the storage archives for complex components like the components of

the provenance framework, along with the Kepler archives that retains all the workflow entities.

Core layer – The core layer contains the basic Ptolemy package and the primary level adaptations by Kepler. It comprises of the core Java APIs and packages used to build the workflow entities.

An actor is the atomic unit of Kepler. It is responsible for performing a task based on the input parameters that it receives from its input ports and it passes on the output to its output port. In a higher-level view of a workflow, the workflow divides the work into different tasks and assigns an actor to perform each of these tasks. These actors are connected to one another to form a sequence defining a scientific process. Figure 6 illustrates the relationship between the workflow and the other underlying resources that perform heavy computations. Figure 9 depicts a job submission actor components residing in different planes. The first part of the figure is the java backend code of the actor that takes an input into a function; the function performs the task and gives an output. This code sits on the control plane (Figure 6) and it internally works with the available resources to decide how to perform the activities to meet the goal. As shown in figure 9, the job submission actor calls the underlying script in order to execute the program that does the job submission task. This script belongs to the execution plane of the computation. It performs the actual task and works with the middleware to access and track the provenance and metadata while the actor just controls the different program invocations.

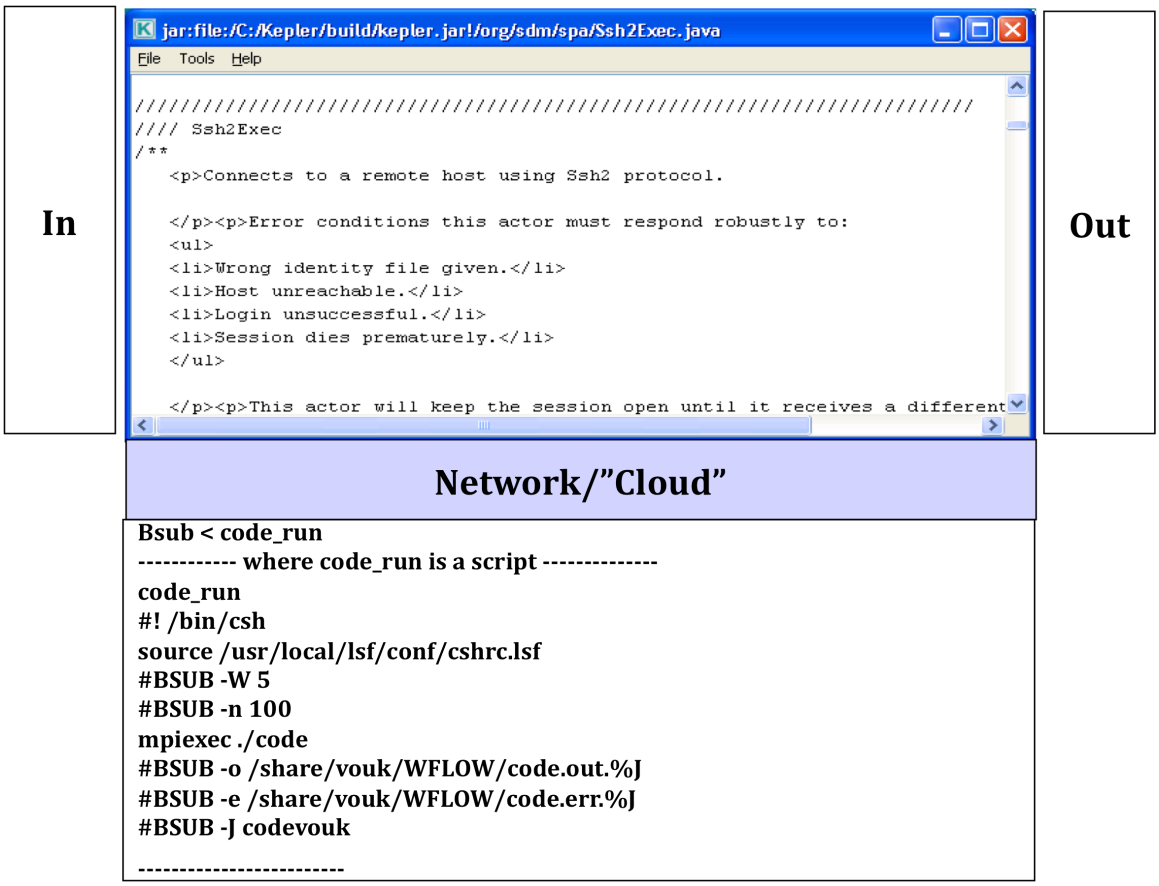


Figure 9 - Actor in a broader sense

4.1.3.2 Features of Kepler

Kepler was launched as a tool for the scientists to develop scientific workflows to create a distributed setting for defining, submitting, executing, monitoring and processing the scientific experiments. The extensibility that Kepler provides is attributed to the following features [Kepler02]

1. Accesses to Scientific Data – The Kepler library contains the ingestion actors like *ReadTable* that can access the data stored in excel files. There are many other actors like *EML2Dataset,* which are used to access and download EML described data sources.

2. Graphical User Interface –GUI is one of the most striking features of Kepler because it is intuitive in nature and the workflow designer gets to visualize the workflow as it evolves. It has a search engine that searches the Kepler library for components, which can be dragged and dropped on the workflow canvas. Workflow is built entirely through the drag-and-drop feature that makes the interface easy to use.

3. Distributed Execution – The Web Service actor and the Grid based actors extend the functionality of distributed execution of the workflows by letting the scientists to use any of the web based component plug-ins. The grid based enables authentication, grid-based data access and third party data transfer services.

4. Prototyping Workflows – Scientists can prototype the workflow before the actual implementation of the workflow using components like composite actors.

5. Searchable Libraries – Kepler has a search engine on the left windowpane that is connected to the library consisting of actors and data sources with numerous re-usable components and data sets.

6. Database Access and Querying – Kepler has database actors like *DBConnect* and *DBQuery* which can connect to the database and access the data sets stored in the database through sql queries.

7. Other Executable Environments – Kepler supports the foreign language interfaces through Java Native Interface through actors like Kepler's Matlab actor, RExpression, Correlation etc., to re-use the existing components and adapt to different computational tools.

8. Data Transformation – Kepler includes a set of data transformation actors like XSLT, Perl etc. for connecting semantically compatible but syntactically incompatible Web services together.

9. Flexible Execution – Workflow execution can be dependent on the interactive user input and Kepler extends the functionality of triggering the workflow execution through the input coming from outside. The *BrowserUI* actor injects the user control and input or output from the legacy applications anywhere in the workflow through the web browser.

10. Configurable Libraries – The users can add their developed components in their local Kepler repository via adding the XML of the component along with supporting manifest file into the Kepler library or through the menu.

4.1.4 Outreach and growth of SPA

At the time of SPA's inception, automating the scientific process through the workflows was its primary focus. However, in addition to developing, deploying and monitoring the workflows for the scientific tasks, SPA team has extended the project to another level by constructing the FIESTA model. This was done to improve utilization of the resources, gain an overall picture, and to ensure the optimal solution to the problem by eliminating redundant factors. Thus SPA's work areas also includes [Crit10];

- Data Provenance
- Fault Tolerance
- Dashboard
- Generic Actors

Provenance and Kepler

A scientific workflow needs is the provenance support. FIESTA provides that. Provenance describes the lineage of the data, workflow structure, run-time processing and data transformations, and Provenance also can be defined as a *collection of meta-data, results or states about an already finished task to enable the result reproducibility, sharing and knowledge reusability in the scientific community* [Davi08].

Effective management of data provenance is extremely important in scientific applications. [e.g. CFS+05] Provenance is at the heart of almost all functionalities, capabilities, and productivity improvements offered by workflow solutions. We distinguish **system**, **workflow**, **process** and **data** provenance categories. In the first category we collect meta-data related to preparation, run-time and post-processing environments – things such as which compiler was used to make the run-time simulation code, possibly pre-processing testing activities, run-time machine characteristics, etc. Workflow provenances is concerned with workflow preparation, history, componentization and templates, and so on. Process provenance offers information about changes in the workflows, and about the order and success of the processes executed on both the workflow control plane (Kepler in our case), and at the sites where remote resource are being used.

Data provenances is often considered the most important of these categories. In our case a lot of it comes from the Kepler provenance recorder, but other sources are used as well. In the case of Kepler, data provenance can be thought of as the complete processing history (transformations, types, etc.) of data products, for example actor identification and invocation parameters, properties such as the time, location, and user of invocation, relevant environment and configuration parameters, etc. This information needs to be persistent and permanently associated with a data product so that its provenance is readily available. It also needs to be searchable, so that data with certain provenance can be easily identified. For example, if a bug is identified and corrected provenance information can help identify which

runs should be repeated, or it can be used in run-time and post-processing phases to quickly access certain information.

Besides providing reproducible results, provenance plays an important role in troubleshooting and optimizing efficiency [Davi07]. SPA has extended Kepler to include a provenance recorder, which has been embedded in the workflow environment in order to give a provenance-recording environment [Moua09]. The large amount of data produced at the end of any scientific workflow is attributed to the repetitive nature of the experiment, which in turn challenges the storage ability of the system. The following figure depicts the database schema used for the provenance database n Kepler.

An example of how provenance can be used to recover workflows is given byimplementation of provenance recording in the Kepler/pPOD system [Bowe08]. A Kepler/pPOD system is a customized version of Kepler scientific workflow system designed specifically for phylogenetic data analysis. The provenance model for this system is unique, in which, the provenance recorder takes the actor's scope into consideration and records information which is relevant to the scope of the actor rather than recording the detailed logs on workflow execution. The workflow is thereby reconstructed and derived from the limited information that has been recorded. The valid state of the data is restored.

Fault-Tolerance and Kepler

Another very important functionality of a fully supportive scientific workflow system is its ability to tolerate errors, fault and failures at run-time. Fault tolerant systems are the *computer systems that handle failures by having a back up procedure in place that masks or recovers from problems so that, from the user perspective there either no visible loss of service or at least the impact is minimized and losses are properly managed* [Moua09, Moua10]. It is desirable for the the self-recovery mechanism to involve minimum or no human intervention. The fault tolerance framework designed for the Kepler consists of a

combination of the information it records as provenance, its check-pointing facility, and most recently its actor-level fault recover ability. Kepler Provenance Recorder (KPR) uses the provenance data to deduce the point of execution failure. The KPR provides support to record the data dependencies and control dependencies between every output produced by every inputThe composite actor called *Checkpoint* provides roll-back and recover-based fault tolerance in Kepler. Workflow errors are typically detected by the user-defined expressions called *port conditions*, described in detail in [Craw08]. This actor enables the user to create checkpoints at various stages of the workflow. The actor handles any failure that occurs within the checkpoint. *Checkpoint* stops the workflow execution in case of a failure and restores to the previous checkpoint. The PR records all this information and the actor switches into the auto-recovery mode. When the *Checkpoint* re-executes the workflow, the provenance database sends all the provenance information to the actor, which eliminates the need for re-executing the entire workflow and rather works on executing the workflow starting from the point of failure where state-recovery was possible. *Port condition expressions are used,* monitor token flows and recognize any irregularity based on the variation of the expected values from the actual ones. Patters and templates can be used to effect fault-tolerance [Dani97].

Dashboard (eSimon)

Scientific Dashboard [Barr07], also part of FIESTA, is a multi-view interface that helps with monitoring and visualization of outputs from scientific computations . The Dashboard system was developed in conjunction with the Center for Plasma Edge Simulation (CPES) [Podh07]. It provides the monitoring platform for the scientists, letting them to interact with the ongoing simulation.

Generic Actors and Templates

Important in the context of the work done in this thesis, the SPA team is currently also developing workflow pattern templates and generic forms of actors to improve the re-

usability and portability of the workflow components. Generic Actors are abstract entities that provide functionality that is common across various domains.

For example, Copying a file, creating an SSH session etc. These actors provide a convenient way for the workflow designers to create workflows without the need to implement the intrinsic details a new. ProcessFileRT [Podh07], developed as a part of the Center for Plasma Fusion Simulation Data (CPES), is a very extensively used generic actor. Another example is SPA's generic data transfer actor [Chin10]. While individual generic actors are a good way of capturing groups of similar operations that can then be invoked from a single actor (akin to a library of functions) by applying different parameters. On the other hand, scientific processes also have sets of tasks (which are not functionally equivalent or similar, that repeat themselves in different applications across various domains. The templates that can reproduce such patterns in Kepler workflows address the issue by systematically encompassing these tasks into a composite actor entity. Composite actors are themselves small workflows that look to the outside world as a single parameterized actor. Underneath, these workflows have their own director and all inputs are governed through the external input/output ports and parameters.

4.2 Case Study Scientific Workflows

Over the years, the SPA team has developed a number of scientific workflows based on the user requirements. In the current work, our focus is on the group of workflows we call monitoring workflows (developed at the Oak Ridge National Laboratory (ORNL). Scientific simulations can run for a long time, hours and sometimes days or weeks over thousands of processors. Such simulations typically produce a significant amount of data. For example, two such simulation codes (discussed in more detail below) called XGC and GTC may produce about 40 GB of data, over 20 hours using over 8000 processors. Once that data is available a scientist may need to do post processing, analytics and job management

operations on the data. For example, transferring the data to a secondary storage for backup. If a failure occurs anywhere along the way, it may mean repeating the whole procedure again. This can waste thousands of CPU hours and a lot of time., Keeping an eye on the simulation and transferring the data as and when they are produced is a tedious manual task that needs the user to be on constant alert. A failure in the simulation makes it even worse. Therefore, casting these manual tasks into automated scientific workflows has been a welcome relief for the scientists from the data management aspect of the experiment, allowing them to focus on the science part of it [Podh07].

Some of the tasks the Center of Plasma Edge Simulation (CPES) automated [Podh07] include monitoring of a simulation for its data output. Transferring the data files to another machine, Converting each data file into HDF5 format., Archiving the HDF5 [HDF510] files to a mass storage system into large chunks, and executing analysis or visualization on the produced data constantly through the various tools to update the user with the current results.

4.2.1 Automation Tasks of the Monitoring Workflows

Based on the analysis done on the Center of Plasma Edge Simulation (CPES), [Podh07] lists the tasks that need to be automated by the workflow. However, many workflows have been developed since, and these tasks are common amongst them as well. Therefore, we list the following tasks as the automation tasks performed by the scientific workflows.

1. Monitoring a simulation for its data output.
2. Transferring the data files to another machine.
3. Converting each data file into HDF5 format.
4. Archiving the HDF5 [HDF510] files to a mass storage system into large chunks.
5. Executing analysis or visualization on the produced data constantly through the various tools to update the user with the current results.

4.2.2 Features of a Monitoring Workflow

Based on the speed, time and processing resources utilized, a typical monitoring workflow developed at ORNL [Podh07] needs to have–

1. ***Security*** – institution-level security requirements need to be honored, and when accessing resources at ORNL, the workflow authentication and authorization mechanisms need to accommodate one-time password, SSH access, certificates and similar parameters.
2. ***Configurability*** – The workflows need to be reasonably re-configurable through parameters (rather than re-write) to allow for changes in resource characteristics, storage paths, etc. It is important that all the resources are compatible with each other or are configurable.
3. ***Robustness*** – In case of failure, it is expected that a workflow will attempt and success in recovering from the failure. This may involve, run-time failure masking, or smart re-run feature in combination with provenance recorder, etc.. For instance, on case of failure, the workflow would resume its state by keeping a record of the finished jobs, unfinished jobs and the questionable jobs. It has the ability to repeat only those jobs the execution of which was interrupted thus maintaining the lowest redundancy rate.
4. ***On-the-fly runs*** – The amount of data produced as a result of the simulation is often too vast to wait for the complete simulation to finish. Instead, scientific workflows transfer the data files as and when they are produced. This minimizes the risk of losing everything in the case of a failure.
5. ***Pipes*** – The monitoring workflows at ORNL use pipelines to keep up with the speed of data generation.

Simulation results are stored in the HDF5 [HDF510] format. HDF5 is a data model, library and file format for storing and managing data designed for flexible and efficient I/O

for high volumes of complex data. It has been observed, from the execution of the XGC workflows, that 20 percent of simulation time is spent in converting the data into this format. In order to speed up the I/O, the monitoring workflows write the data in a special binary format. This format is quick and efficient. These files are then transferred onto post-processing machines. The conversion of the file into HDF5 format is then carried out in the secondary storage system and on the post-processing machines. This is less expensive and brings about a drastic improvement in the overall performance.

Following are the different monitoring workflows, developed at ORNL, that were used to explore the concept of automatic workflow generation.

4.2.3 XGC Simulation Code

The Center for Plasma Edge Simulation Project (CPES) is a part of the Fusion Simulation Projects funded by the DOE. CPES has created an integrated predictive plasma edge simulation code package called XGC to understand localized mode instabilities which reduce the confinement properties of the plasma and shorten the wall's lifetime [Podh07, Klas06]. XGC [Ku06] consists of a set of two different particle simulation codes (XGC-0 and XGC-1). XGC-1 is a full-f gyrokinetic ion-electron particle code derived from XGC-0, designed for integrated simulation of neoclassical and electrostatic turbulence physics using enormous parallel processors.

From the perspective of the current project, the most interesting requirement of the XGC project is the ability to access, monitor and control the application during its execution as it runs in a distributed/parallel environment. XGC workflow was introduced as a solution for managing and controlling this simulation. Appendix A contains a snapshot of the XGC Workflow as developed by ORNL.

4.2.4 GEM Simulation Code

Gyrokinetic Electromagnetic Turbulence Simulation (GEM) is another simulation code conducted and it focuses, as the name implies, on gyrokinetic particle simulation of turbulent transport in burning plasmas [Lee07]. It is a delta-f particle code consisting of the dynamics of the gyrokinetic ions and drift-kinetic electrons. The code accurately measures the magnetic fluctuations by studying the well-magnetized plasma physics. The accuracy and precision of the fluctuation levels comes at the cost of large amount of data being produced and stored. ORNL developed a GEM monitoring workflow for the data management of this simulation. Appendix B contains a snapshot of the GEM workflow.

4.2.5 Pixie3D Simulation Code

Parallel, Implicit, eXtended MHD 3D Code (Pixie3D) [Chac04] is a parallel and fully implicit 3D extended MHD solver. The goal of the Pixie3D project is to demonstrate the path for fully implicit MHD in general geometries using state of art scalable solver technology and exploiting massively parallel computing environments. Appendix C contains a snapshot of the Pixie3D workflow.

4.2.6 Analysis of the Monitoring Workflows

All simulation codes of interest here deal with a lot of data. The accuracy and precision achieved by these packages come at a cost of recording and maintaining large amounts of output data, which need to be moved to a more permanent storage as soon as they are produced. Monitoring workflows automate the process of data movement management through a user-friendly interface. making this procedure of complex steps simple.

Appendices A, B and C show the three monitoring workflows developed at ORNL, Inspection of the three workflows shows some common set of steps. For example,

establishing the SSH connection with the remote machine, creating output directories, waiting for the simulation to finish its execution and produce the data files, thereafter splitting the file, transferring the file, appending the file, plotting graphs, text and image archival and making movies. The automation tasks contribute to many repeated operations performed on individual data files where the simulation produces data in time steps [Podh07]. It turns out that most of the monitoring workflows used to monitor an ongoing simulation, and do the post processing data management, follow similar steps. The goal is to determine the pattern followed by the monitoring workflows, use that to develop executable templates to help automate the process of constructing such workflows through parameterization.

Monitoring workflows have an extra visualization step where it is left to the user's discretion whether to have the data produced plotted or not. The XGC workflow as showed in Appendix A uses AVS Express for visualization. On the other hand Gem data are typically plotted using VisIT and that step is not part of the workflow.

Recently, we developed a *client-server* model to plot data using command-line VisIT [Viz10]. The server has VisIT program running on it while the client invokes the data plotting method by passing the data through the commands. The idea is to have a workflow actor execute the client script once it has the data. The image plotted from the Pixie3D data is shown in Appendix E. This *client-server* visualization model is yet to be tested on the existing workflows. Although, visualization is out of the scope of this thesis, the future expansion would include a method that generates a component for plotting data into images.

Chapter 5 - Workflow Templates and Patterns

In software engineering, a design pattern [Gamm95] is defined as "*a reusable abstract solution for, or an approach to the solution of, different variations of frequently occurring problems*." A design pattern is structure oriented and distinctly defines the entities and the relationships among them. A template [Barr94] is "*a reusable solution that focuses on the explicit details of the solution through parameterization*". It is a canonical but explicit solution to a specific problem, based on a pattern, that can be refined, customized and modified to suit different nuances of the same problem.

A workflow design pattern is a model that is reusable for creating workflows using relevant design pattern primitives. Business process workflows often follow the control flow model, while scientific workflows often follow the data flow model. Furthermore, business workflows tend to be pattern and template driven, while scientific workflows are often one-of creations. However, a pattern (and template) for a scientific workflow would mean combining the notion of both models in order to identify a generic pattern with a high re-usability factor.

Design patterns range from simple constructs to complex structures depending on the context they are used in [Aals03, Yild09]. Every workflow is an enactment of certain business or scientific processes, which it specifies, executes and monitors. A workflow process definition specifies the activity expected to be performed by the workflow, and the order in which they need to be performed. Workflow patterns are context oriented. That is, the workflow specifies a set of actions performed in a specific context. The semantics of a workflow is clear in its context but not necessarily so outside the context. These patterns are applicable solutions only if the workflow requirements fall into the specific context the patterns belong to.

Monitoring workflows define one such context. In that context the workflow performs a specific set of steps involving post processing of the data that is produced by, for example, a running simulation. Data patterns [Russ05] are the constructs that capture the various ways in which data is represented and utilized in workflows. They are most effective when used for the development of scientific workflows. There are different characteristics that the scientific workflow repeatedly exhibits [Russ05]. This includes:

1. *Data visibility,* which relates the definition of the workflow construct to the scope of the workflow making them accessible to one and all the workflow components that might need them.
2. *Data Interaction* that relates to the mode of data communication among the different components of the workflow.
3. *Data Transfer* specifies the different ways that the data gets passed or transferred from and to the different components within and outside the workflow. Data based routing characterizes the way in which the data can influence the aspects of the workflow, i.e. the control aspect of the workflow taken care of by the data.

In this section we describe some of the basic design patterns that we have identified and used for the monitoring workflows. We describe our approach to identifying the patterns in these workflows, and translating them into re-usable templates, as well as a pre-existing "atomic" template called ProcessFileRT, and the templates built for our research.

5.1 Basic Workflow Patterns

Most of the design patterns used for the scientific workflows are the Data Patterns involving one or more operations performed on the data. Scientific workflows are complex structures, but for most part they are composed of the basic workflow structures shown in

Figure 10. In [Bhar08], the authors characterize the scientific workflows by defining the basic workflow structures, specifically data-flow structures.

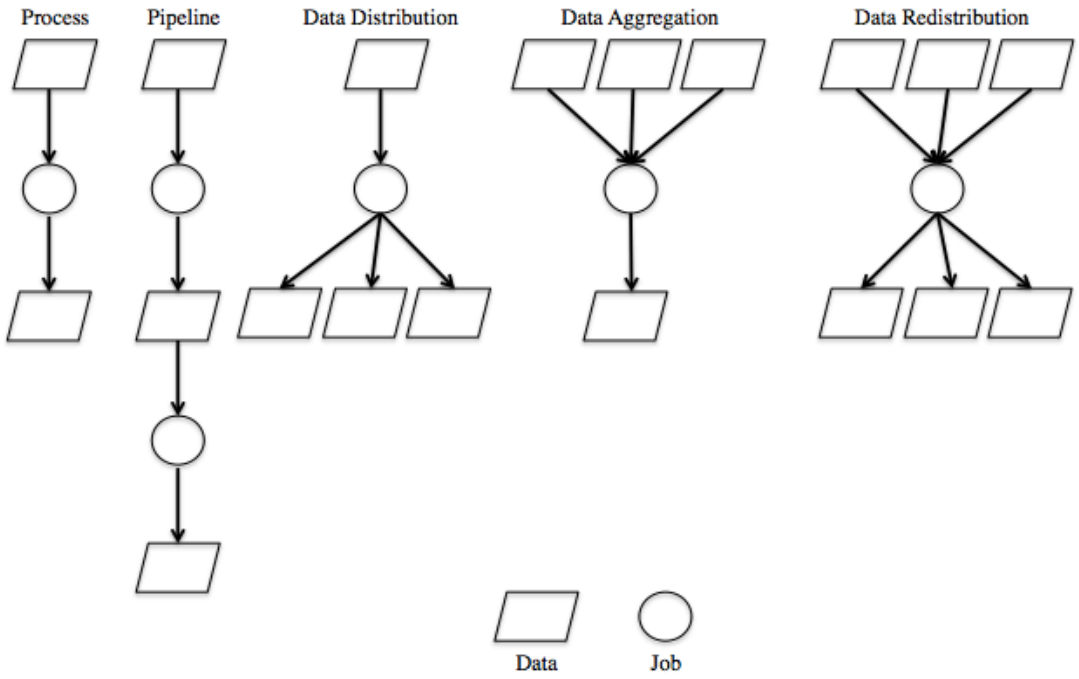


Figure 10 – Some basic Workflow Patterns [Bhar08].

Process pattern is a simple pattern that takes in the input data and outputs the data product. *Pipeline* pattern consists of several such data processing jobs connected to one another forming a pattern in which an output data product of one job becomes the input data of another job. *Data distribution* pattern is the most commonly used pattern in the SDM's monitoring workflows. Depending on the context, the job produces data in form of chunks to serve either or both of the two purposes; data chunks serve as an input to several jobs or the job produce data chunks as an efficient data management technique for large amounts of data. Data distribution plays an important role in rendering parallelism to the workflow. *Data aggregation* is a pattern that takes in data from multiple sources and produces a single data product. Although this pattern does not exactly ensure parallelism but when combined with

the data distribution structure, it becomes a powerful tool, which is the *Data redistribution* pattern [Bhar08].

A large number of complex linear scientific workflows can be built using these basics structures. Of course, if there is a feedback loop, that requires a more complex pattern. Similarly, processing decision points may require additional patterns. The different data patterns defined in [Russ05], the patterns defined in [Gome08] and the automation tasks of the monitoring workflow described in the previous section are the complex compositions of these basic structures.

5.2 CPES Generic Template – ProcessFileRT actor

A major part of all the monitoring workflows developed by ORNL deal with operations that involve SSH sessions with remote machines. They are gateways to remote scripts and codes where they demand programs that can execute on remote sites whilst maintaining the same session for multiple program execution. Kepler has an *SSHExec* actor that uses the org.kepler.ssh java package so that it shares a common session among multiple executions [Podh07]. The command execution carried out by the *SSHExec* actor is an atomic step in the overall workflow execution, it takes in the command as a string, executes it and outputs stdout, stderr and errorcode.

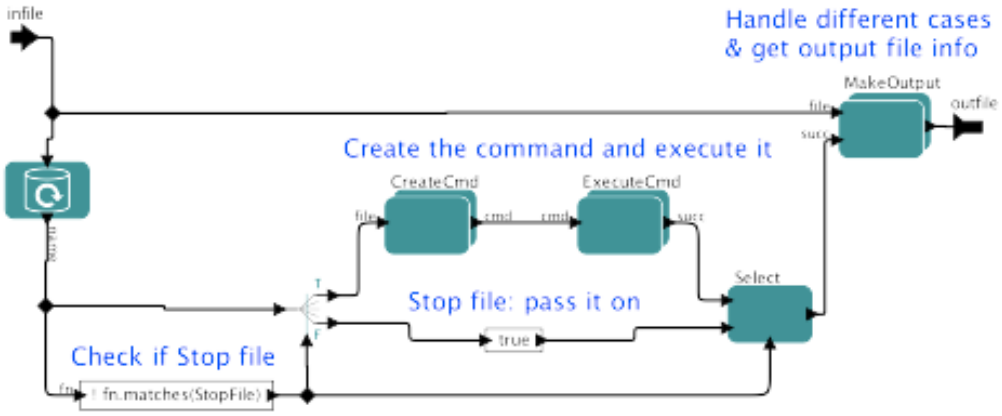


Figure 11 – Top-level view of the ProcessFile actor [Podh07]

For the CPES[2] project, the team created a generic actor called the *ProcessFile* actor [Podh07]. This actor is itself a workflow that inputs a file, executes the program on the input file and generates an output file. The significance of this actor is that it encapsulates a checkpointing facility for successfully executed program files, and discards the tokens that are related to the failure. It also includes a logging functionality that logs the success and failure reports. Another relevant feature of this actor is its ability to identify the termination of the data products that come via its input port through the termination file called the *stop file*. Figure 11 shows the high level view of the *ProcessFile* actor.

As shown in Figure 11, the *CreateCmd* composite actor reads the input file and creates an executable command. The *ExecuteCmd* actor executes the command and takes care of checkpointing and logging. *MakeOutput* actor is responsible for creating the output file and handling an error.

[2] Center for Plasma Edge Simulation

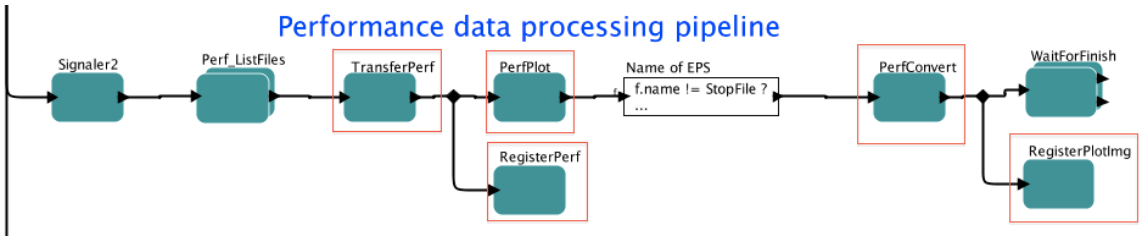


Figure 12 – The Performance Processing pipeline of the GEM monitoring workflow. Red boxes indicate the ProcessFile actor

ProcessFile actor is the most widely used actor in building the workflows we studied because of its flexibility and extensibility in performing a variety of functions like file transfer, file conversion, image creation operations. Figure 12 is the snapshot of the performance-processing pipeline taken from the GEM monitoring workflow. The red boxes highlight the *ProcessFile* actors. More than 50% of the actors used in this pipeline are the *ProcessFile* actors. The actor has also been used as an atomic element in text and image archival in monitoring workflows. It hides the complexity, related to all the features like checkpointing and logging from the user, letting the user to parameterize the actor. The parameter set for the *ProcessFile* actor, as shown in Appendix D, is the user's view of the actor. As will be described in the next chapter, the generality of the *ProcessFile* actor that serves as a template has been a major advantage in creating the workflows automatically.

5.3 Our approach to templates

The primary focus of this thesis is to analyze, examine and assess the possibility of generating scientific workflows through parameterization using templates and design patterns. We can classify our approach in the following three steps –

1. Creating a library of templates that are built based on our analysis of commonly encountered design patterns in the reviewed monitoring workflows.

2. Developing a sample tool that intelligently uses the created temples from the library and adds user specified dependencies among these templates resulting into a fully functional workflow.
3. Testing the tool by generating monitoring workflows other than the three workflows used as a reference and comparing them with the original workflow.

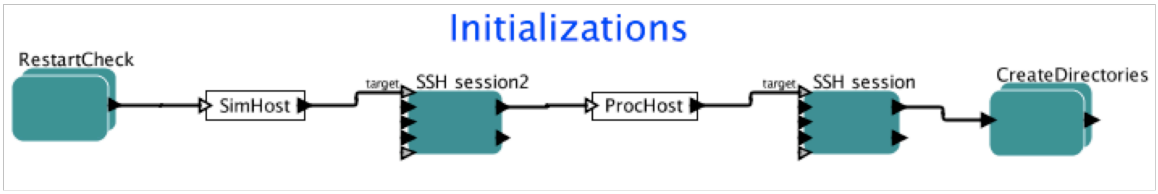
The first step was to develop templates that can be reused across different monitoring workflows. This chapter contains the details of template development. These templates are not only meant for the workflow generator wizard, but they can be easily accessible for any workflows that are created with or without using the tool. They can be made available to all the users by adding them to the Kepler library. Our technique of template creation is somewhat similar to [Gome08], we encapsulate the atomic components of a workflow (actors) based on their collective functionality. Also, we divide the existing workflows based on the individual or collective functionalities and define templates by encapsulating the functionality that is identical and unchanged in all the reference workflows, into composite actors. The next segment details each of the templates created for the research.

5.4 Templates for the Workflow Generator Wizard

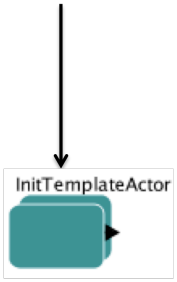
Studying the monitoring workflows in its entirety and comparing them seemed to be an intricate process. We divided the monitoring workflows into different stages to make it easier to compare different workflows stage by stage. Specifically, we divide a monitoring workflow into the following stages –

1. Preparation steps
2. Processing steps
3. Performance monitoring steps (optional)
4. Post-processing steps.

We now present the templates that have been created for each of these stages.



Sub-Workflow Extracted from the GEM workflow



Initialization Template

Figure 13 – Translation of the Preparation Steps of the Monitoring Workflow into a Template

5.4.1 Preparation Steps

The preparation steps include the initializations that are performed to prepare for the execution of the rest of the workflow. This stage creates the SSH sessions (needed for the different remote operations) and the output directories. These are the common steps performed by every monitoring workflow as the initialization process.

Figure 13 shows how the initial workflow preparation steps were translated into a template which is a composite actor entity encompassing these steps. The "*CreateDirectories*" composite actor in the figure depicts the encapsulation of multiple instances of a "*CreateDirectory*" class defined in each of the monitoring workflows. The composite actor shown in Figure 13 creates multiple output directories for storing the text, images, transferred files, etc., produced as an output. Besides creating a template that is convenient and easy to understand, our aim was to keep the template definition as simple as

possible. Therefore, to eliminate the dependency of the template on a separate class entity being referred by its instances in the template, we composed the Initialization template as shown in Figure 14 that functions the same way as the one shown in Figure 13.

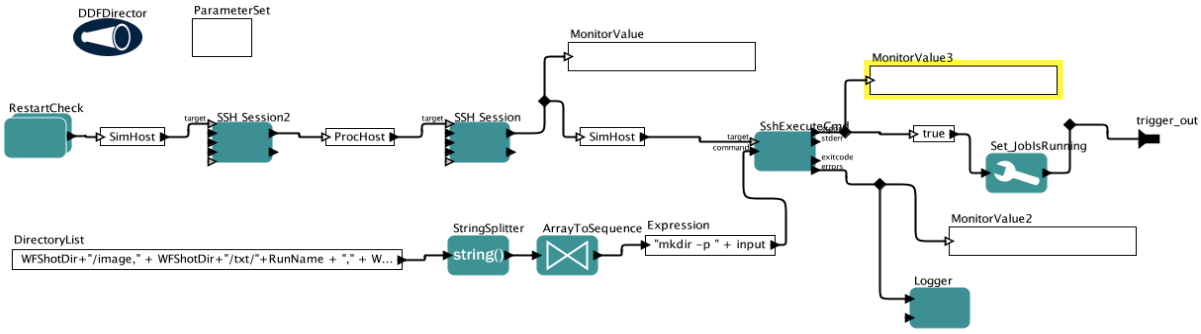


Figure 14 – InitTemplateActor Template

5.4.2 Post-processing Steps

The post-processing steps in a monitoring workflow include the text and image archiving and the movie making. They are composed into two different pipelines, one for text archiving and another for archiving images & making movies. These pipelines also appear to be the commonly used set of steps in a monitoring workflow. Therefore we translated each of these pipelines into a standalone template as shown in the following figures (Figure 15 and 16).

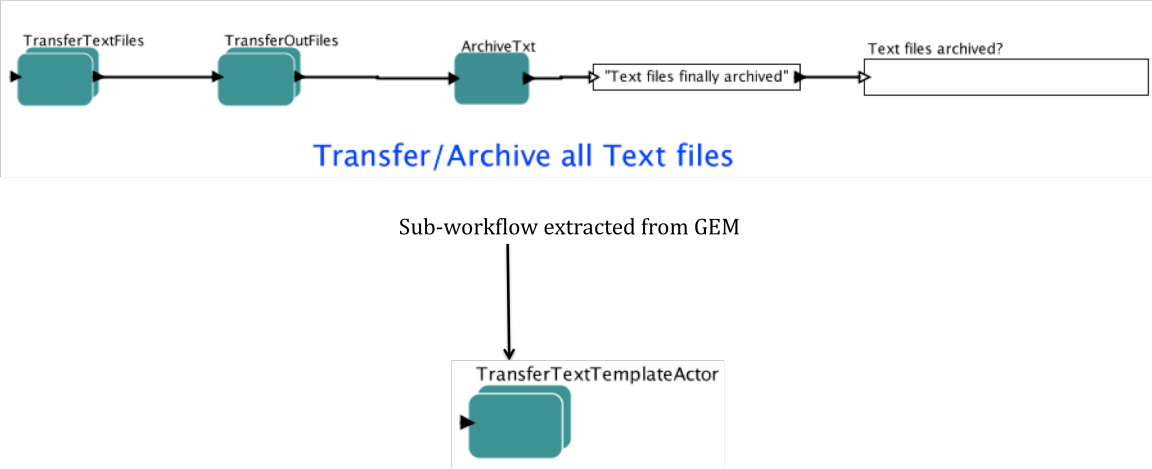


Figure 15 – Translation of the text archival pipeline into the TransferArchiveText template

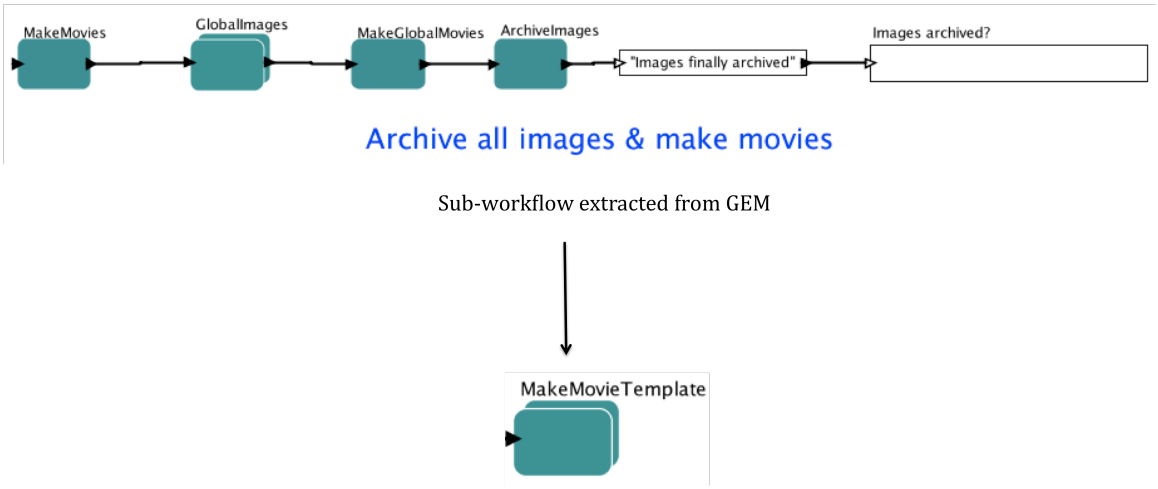


Figure 16 – Translation of the image archival & movie making pipeline into the MakeMovieTemplate template

5.4.3 Performance Monitoring Steps

Performance monitoring is enacted in the monitoring workflows by a dedicated pipeline, as shown in the XGC and Gem workflows (Appendix A & B). It includes transferring the performance data from the host where the simulation is running to the processing host, plotting the data, converting the data files and registering them. These are a standard set of

steps followed to analyze the performance of the application. However, as will be discussed in the next chapter, for flexibility and usability we choose to translate just a segment of the pipeline, as shown in Figure 17, into a template instead of converting the entire pipeline into a template. The idea behind it is to maintain the generality of the workflow generator wizard that produces two actors (Signaler and ListFiles), that are the starting points of the pipeline based on the user input.

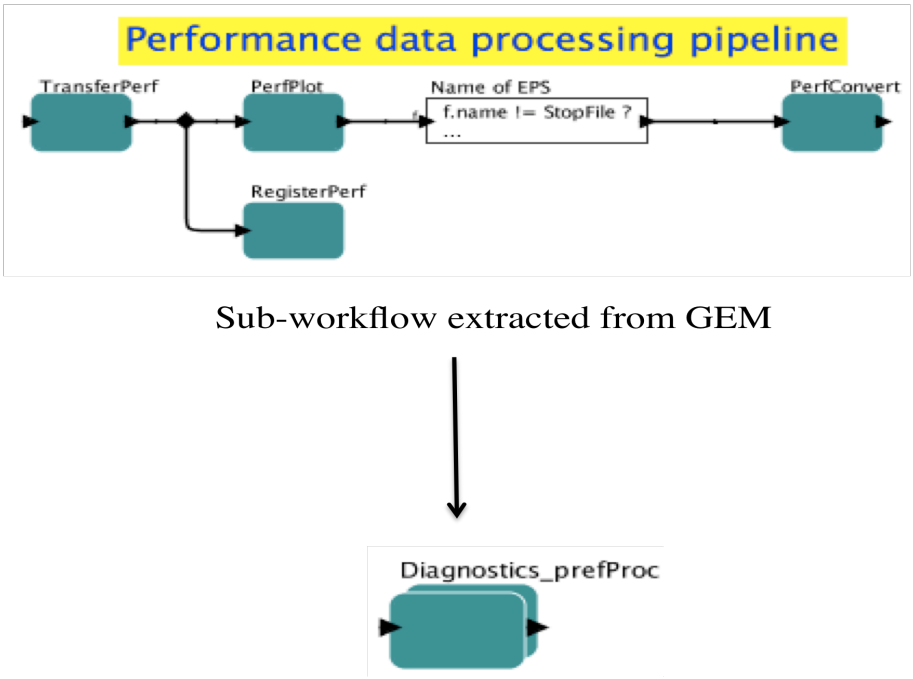


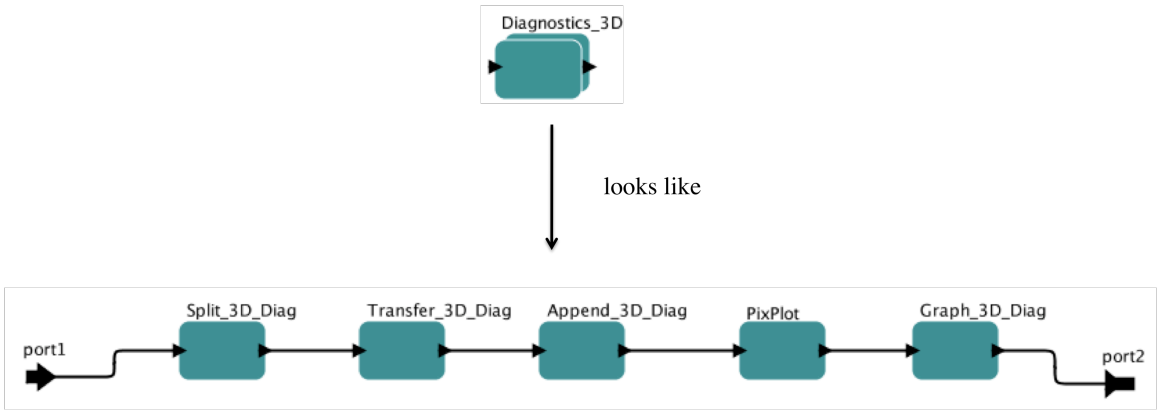
Figure 17 – Translation of Performance Processing pipeline into a template

5.4.4 Processing Steps (Auto-generated Template)

Processing steps are translated into templates in a slightly different way than the steps. The processing pipeline typically consists of functions such as splitting the file, transferring the files to the processing host, appending the file and plotting the data into a graph. This seems to be a straightforward set of actions as seen in the 1-D processing pipeline of the

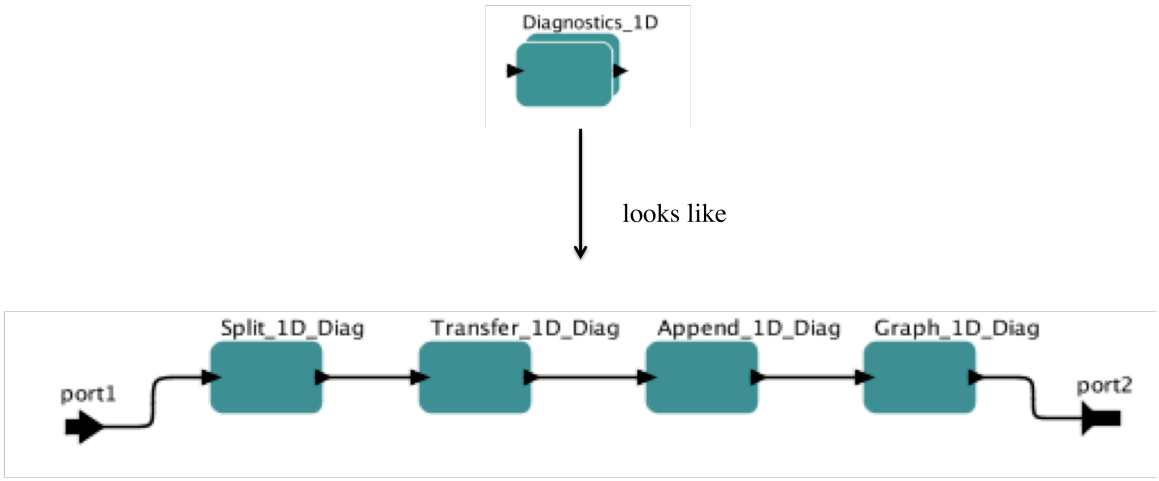
XGC workflow (Appendix A) and the 2-D processing pipeline of the Gem workflow (Appendix B). However, the 3-D processing pipeline of the Pixie3D workflow (Appendix C) and the 0-D processing pipeline of the Gem workflow are slightly different. The former has an extra step called *PixPlot* while the latter does not perform the split and the append steps. Based on these minor differences in these processing pipelines, if we create three different templates, one for each of the cases discussed, we would end up creating a lot of redundant templates. If we create just one template, which carries out the four basic steps of split, transfer, append and graph as seen in the 1-D and 2-D pipelines, then the template is not usable for the 0-D and 3-D pipelines. The user would have to either create them from the scratch, or add the extra step and change the dependencies accordingly or discard the unwanted steps and arrange the dependencies. This extra work on the user's part contradicts our purpose of using templates.

In order to overcome this issue and arrive at a more generic solution for each of these pipelines, we generate processing workflow on the fly instead of pre-defining a template. On examining each of these pipelines, we discovered that every step in each of the processing pipelines is performed by passing a command to the *ProcessFile* actor, discussed in the previous section. That is, each of these steps in every pipeline is performed by passing the appropriate parameters, as shown in Appendix D, to the *ProcessFile* actor. All parameters get their values from the user input. Therefore, the workflow generation wizard (discussed in the next chapter) decides which template to use where, and determines the type and the number of processing pipelines needed in the workflow, as well as the steps to be performed in each of these pipelines based on the user input. The system constructs the workflow pipeline-by-pipeline, determines the steps to be performed in each pipeline and encapsulates all these steps into a single entity. For example, Figure 18 shows the auto-generated template for the 3-D diagnostics pipeline of the Pixie3D workflow (Appendix C) and Figure 19 shows the auto-generated template for the 1-D diagnostics pipeline of the XGC workflow (Appendix 1).



Auto-generated 3D diagnostics for the Pixie3D workflow specifications.

Figure 18 – Auto-generated template, Diagnostics_3D for the 3-D pipeline of the Pixie3D workflow



Auto-generated 1D diagnostics for the XGC workflow specifications.

Figure 19 – Auto-generated template, Diagnostics_1D for the 1-D pipeline of the XGC workflow

5.4.5 Miscellaneous Templates/Generic Actors

There are certain components that exist in the original workflows that act as a template or generic actors on their own. Although, these components are not added to the public version of the Kepler library, instances of such templates are used in all monitoring workflows that

have been considered so far. We now list all such existing templates/generic actors in the following table.

Table 1 – Existing Templates/Generic actors in the Monitoring Workflows

Template	Brief Description
	JobWatcher checks whether or not the simulation is under execution and signals when it stops.
	The Signaler is a python script that triggers the execution of the pipeline that is starts from it.
	ListFiles_D composite actor comprises of the SSHDirectory listing actor for listing all the directories.
	WaitForFinish waits for the “finished.sim” which is the signal that the pipeline has finished executing. This actor is used to synchronize the execution of all the pipelines.
	ProcessFile is a gateway to remote scripts and codes where they demand programs that can execute on remote sites whilst maintaining the same session for multiple program execution.

Chapter 6 - Workflow Generator Wizard

Kepler-based scientific workflows are fully formed MoML files (written in XML). Kepler graphical user interface (GUI) provides a graphical (and graph-based) representation of this XML file. Every drag and drop activity, or any change/additions in the dependency between actors, triggers a change in the XML file. Out of the three monitoring workflows we analyzed, XGC was developed first. After that came Gem, and Pixie3D followed. While developing Gem, XGC was used as the reference workflow and Gem was used as the reference workflow for developing Pixie3D. That did provide some inherent similarity. The process of building a workflow based on an already existing workflow suggests that there are two ways to build such as workflow,

1. Make a copy of the existing workflow and adapt it by adding or deleting some components and links as per the workflow requirements.
2. Copy the components that are expected to be the same in the new workflow and then manually construct the new workflow.

If the new workflow has more than 50% of components that are in common with an existing workflow, the developer might want to choose the first approach to creating the new workflow, and the second one otherwise. Although *ProcessFile* is the widely used generic actor, and majority of the workflow entities in our cases are instances of the *ProcessFile* actor, tweaking an existing workflow would still mean adding these actors and parameterizing them.

The graphical representation of the workflow depends on the workflow designer, which indirectly affects the prospective workflows to be developed based on it. For example, the three production workflows discussed in this thesis have been developed by the same

developer. As a result, they share a lot of similarities. All workflows that are constructed based on these three workflows would then follow the similar protocol. The process enacted by the workflows becomes easy to understand if the similar workflows have a uniform design and representation. It is therefore of interest to design a tool for constructing complex workflows automatically so that all similar workflows follow the same protocol in terms of the order of steps and the graphical representation.

6.1 An XML Representation of a Workflow

The Workflow Generator Wizard is responsible for assembling the workflow components intelligently and producing a workflow that satisfies all the user requirements. As already mentioned, Kepler workflows are described using an XML-based language called MoML [Lee00]. The way our automatic workflow generator "wizard" works is that every component in the workflow is treated as an *XML snippet* by the generator. It is the job of the wizard to identify all the relevant snippets and put them together. Figure 20 is an example of the XML view of the *PN Director*[4] and the *Logger* actor.

The properties of a component are maintained in the XML structure through the well-formed XML tags. Every XML tag has three identity variables – *name, value* and *class*. *name* identifies the name of the sub-component, *value* identifies the value of the variable and *class* denotes the class to which the sub-component belongs. Every tag is contained in some parent component, and the outermost parent that contains all the tags in the file is the <*xml*> tag. The <*property*> tag denotes some property of the parent container that contains the tag. The <*entity*> tag denotes a container that contains some component by itself. The <*port*> tag represents an input or output port of the component that it belongs to. These are the three

[4] Kepler processes (actors) are controlled by process engine called Director. Kepler supports a number of Directors. In most of the workflows of interest two are used – SDF – Sequential Data Flow Director, and PN or Process Network Director. The latter allows for parallel execution of the workflow or workflow parts it controls.

major MoML components encountered in a workflow. There are many other tags that can be spotted in a workflow's XML representation such as <*configure*>, <*relation*> and <*link*> to name a few.

```
<property name="PN Director" class="ptolemy.domains.pn.kernel.PNDirector">
    <property name="initialQueueCapacity" class="ptolemy.data.expr.Parameter"
        value="5000">
    </property>
    <property name="_location" class="ptolemy.kernel.util.Location"
        value="[275.0, -115.0]">
    </property>
</property>
```

```
<entity name="ErrorLog" class="org.kepler.actor.Logger">
    <property name="logfile" class="ptolemy.data.expr.FileParameter"
        value="$ErrorLogFile">
    </property>
    <property name="header" class="ptolemy.data.expr.StringParameter"
        value="$LogHeader">
    </property>
    <property name="format" class="ptolemy.data.expr.StringParameter"
        value="$LogFormat">
    </property>
    <property name="entityId" class="org.kepler.moml.NamedObjId"
        value="urn:lsid:kepler-project.org:actor:509:1">
    </property>
    <property name="class" class="ptolemy.kernel.util.StringAttribute"
        value="org.kepler.actor.Logger">
        <property name="id" class="ptolemy.kernel.util.StringAttribute"
            value="urn:lsid:kepler-project.org:class:1209:1">
        </property>
    </property>
    <property name="_location" class="ptolemy.kernel.util.Location"
        value="{545.0, 345.0}">
    </property>
    <port name="text" class="ptolemy.actor.TypedIOPort">
        <property name="input" />
        <property name="_showName" class="ptolemy.data.expr.Parameter"
            value="false">
        </property>
    </port>
    <port name="logfile" class="ptolemy.actor.TypedIOPort">
        <property name="input" />
    </port>
</entity>
```

Figure 20 – MoML view of the PN Director and the Logger actor

6.2 Architecture

The Workflow Generator Wizard consists three main components (Figure 21) –

- ***Web Form*** – The web form is the only interface of the system that is accessible to the scientist or the workflow engineer. This form can be any javascript enabled HTML based form that has a *Submit* button. It is to prompt the user to enter the mandatory parameters like the remote machine address, the simulation host address, etc. The user would be asked to choose between the various available options like 1-D processing, 2-D processing, performance processing etc. Based on the options chosen, the user is prompted for the relevant parameter values that are needed to build a workflow on demand. Once the user enters all the required fields and clicks the *Submit* button, the *main* method of the generator is triggered which builds the workflow and sends back the workflow XML file to the user. One thing to note is that in the current proof-of-concept, we have not yet implemented this web interface, but assume that the data will be transmitted to the generator through such an interface.

- ***Library*** – It is a collection of all the actors and templates stored in form of *xml snippets* discussed in the previous section. The existing Kepler library can serve as a repository to store the templates along with the actors. Typically that is done by having project or domain specific sub-libraries with relevant actors, templates and composite actors.

- ***Generator*** – Generator is the central entity that acts as an arbitrator between the library and the user interface. It is the controller that adjudicates the flow of action based on the user input communicates with the library and intelligently assembles the *xml snippets* together generating an XML file that represents the workflow of interest. Human verification of the outcome of the generation is highly recommended.

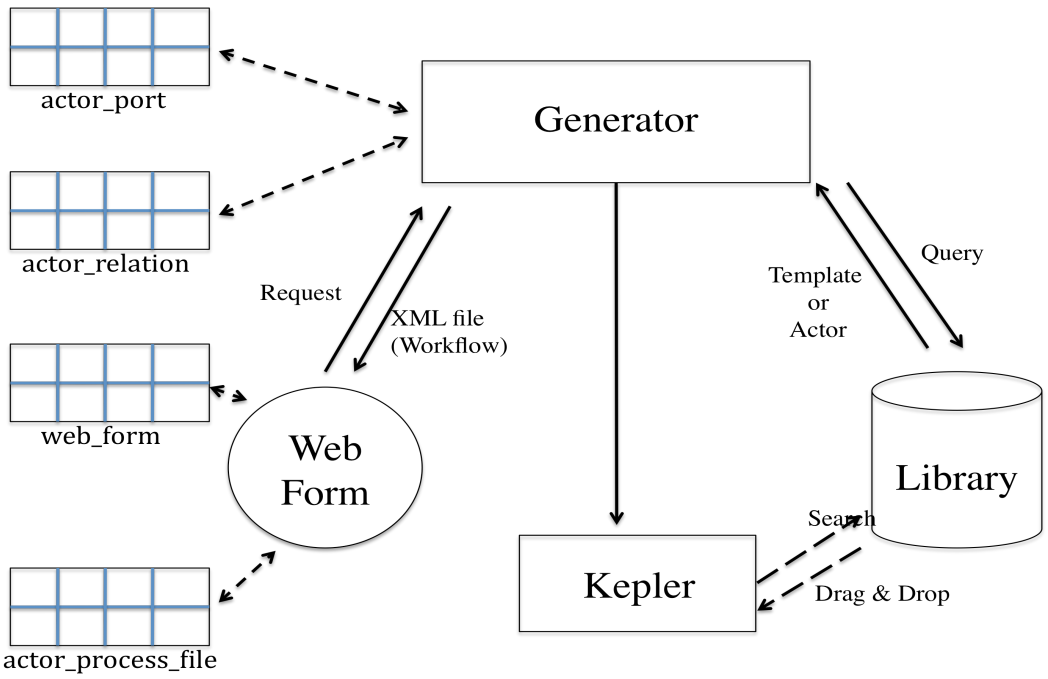


Figure 21 –Architecture of Workflow Generator Wizard

6.3 Implementation

Our proof-of-concept (POC) implementation instantiates the architecture only in part. The primary working component is the generator, while the web interface and the Kepler library are currently simulated. As the project progresses towards production implementation, both will be either developed (GUI) or linked in (Kepler libraries). The *Workflow Generator Wizard* is implemented in Java. Implementing the generator in the same technical environment as the other projects was to make it easier to integrate with the existing SPA system in the future. As already mentioned, the *Workflow Generator Wizard* that has been implemented does not include the GUI. We have developed the Generator module and we tested it by simulating the library and the web form modules through tables described below.

6.3.1 Web Form

The *Web Form* module represents a set of web pages that acts as a user interface and is dynamic in nature with run time behavioral properties. We simulate the module using the following two tables. For the testing purposes, the *Generator* module takes the information from these tables instead of the web pages.

- ***web_form* table** – This table stores the different options that are to be given to the users and the parameters for each of these options. A '0' or a '-1' in their identifier column indicates whether the option has been chosen or not. The option id in the identifier indicates the option to which the parameter belongs. For example, *A '0' in the identifier for the name '3-D Diagnostics' with an id '3' (say), indicates that it is not a parameter, but an option that is given to the user and the user has chosen the workflow to have a 3-D Diagnostics pipeline. The parameter 'PixPlot' with an identifier value '3' means the parameter is required to have a value if the user chooses the workflow to have a 3-D pipeline (since the identifier of the 3-D Diagnostics is '3'*). So in actual implementation of the *web pages*, the generator would access this table twice, once to populate the page with the different available options and update the table. Second time to access all the parameters that are needed for deploying the option.

```
mysql> show columns from web_form;
+-----------+-----------+------+-----+---------+-------+
| Field     | Type      | Null | Key | Default | Extra |
+-----------+-----------+------+-----+---------+-------+
| id        | int(11)   | NO   | PRI | 0       |       |
| parameter | char(100) | YES  |     | NULL    |       |
| status    | int(11)   | YES  |     | NULL    |       |
| value     | char(200) | YES  |     | NULL    |       |
+-----------+-----------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Figure 22 - The web form table

- ***actor_process_file* table** – The workflow that is generated by the system can also act as a "*fill in the blank*" model allowing the user to make changes to the workflow by adding new actors and changing the existing dependencies. As discussed in Section 5.2, *ProcessFile* actor is a widely used component in a workflow that is multipurpose for performing most of the processing steps. In order to create the auto-generated template, the user is prompted for action to determine whether he wishes to customize the processing steps by adding or deleting any of the processing steps to be performed by the instances of the *ProcessFile* actor. The information is stored in this table in a *parameter versus value* format for all the parameters of the actor shown in Appendix D. The generator accesses this table while constructing the pipeline of processing steps. The actor described by a single table entry has an attribute called pipeline that tells us the pipeline that the actor belongs to.

```
mysql> show columns from actor_process_file;
+------------------+----------+------+-----+---------+----------------+
| Field            | Type     | Null | Key | Default | Extra          |
+------------------+----------+------+-----+---------+----------------+
| id               | int(11)  | NO   | PRI | NULL    | auto_increment |
| actor_name       | char(50) | NO   |     | NULL    |                |
| pipeline         | char(15) | YES  |     | NULL    |                |
| RemoteMachine    | char(50) | YES  |     | NULL    |                |
| Command          | char(50) | YES  |     | NULL    |                |
| InDir            | char(50) | YES  |     | NULL    |                |
| OutRemoteMachine | char(50) | YES  |     | NULL    |                |
| OutDir           | char(50) | YES  |     | NULL    |                |
| Ext              | char(50) | YES  |     | NULL    |                |
| ReplaceExt       | int(11)  | YES  |     | NULL    |                |
| doProcessing     | int(11)  | YES  |     | NULL    |                |
| doCkpt           | int(11)  | YES  |     | NULL    |                |
| doCheckOutput    | int(11)  | YES  |     | NULL    |                |
+------------------+----------+------+-----+---------+----------------+
13 rows in set (0.42 sec)
```

Figure 23 - The actor_process_file table

6.3.2 Library

All the templates and the components are stored in form of text files. A variation would be storing them as XML files. We have done testing by storing the templates in the Kepler library and accessing it by conventional *drag & drop* method. However, in the current POC implementation, accessing the library means reading and writing files.

6.3.3 Generator

This module includes a set of Java files and two tables. The *actor_port* table is a temporary table maintained strictly for simulation purposes. It maintains a list of ports in each of the actors in the workflow. In order to give a proof of our proposal, our aim was to re-create the three reference workflows using the *Workflow Generator* Wizard and compare with the original ones. So it was important to maintain the state of each actor, and every actor had the input/output ports named differently. Hence, the ***actor_port*** table was maintained to refer to, every time an instance of the actor was to be created. The second table is the ***actor_relation*** table that maintains the dependency that binds an actor through its ports. As observed in most of the workflows, an actor can be connected to more than one actor and in such a case all the three actors share a common relation. Therefore, it becomes very important for the system to remember the relation between the first two actors to be used while creating a dependency for the third actor. This table maintains all such actor port to relation name entries. The table is updated whenever a new dependency is introduced.

The *Generator* uses a number of helper classes and a controller class to create the workflow. These classes internally access the library and the web form tables. Based on the user input contained in the *web_form* table, the generator creates the pipeline and assigns a *Signaler* and a *LisFiles* actor to each of them. Figure 24 is a snapshot of the Pixie3D workflow produced by the *Generator*.

6.3.4 Performance Analysis and Testing

The main intent of making the task workflow construction easier for the scientist is fulfilled because all that the scientist needs to do in order to build the workflow using the wizard is to give the workflow parameters while choosing the right options and the wizard generates the workflow. This is a more convenient solution for generating a workflow that is similar to an existing workflow instead of constructing one from the scratch.

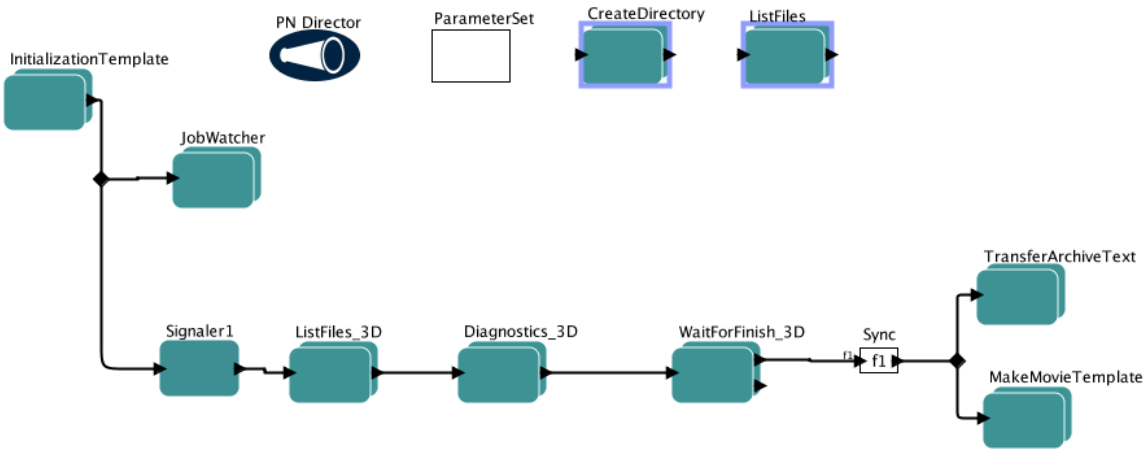


Figure 24 - Auto-generated Pixie3D Workflow

Figure 24 is the Pixie3D workflow that is generated by the wizard. Comparing with the original Pixie3D workflow shown in Appendix C, drastic considerable decrease in the number of user-level actors in the workflow can be noticed. On an average, a scientific workflow built manually has 22-25 actors on the higher-level with many more underlying actors. User is responsible for all of them. Although, all the actors and the respective relations that are seen in the original workflow do exist in the auto generated workflow also, the wizard hides the underlying sub-workflows from the user by encompassing the complexity into a single actor and makes the high level definition of the workflow much more simple with no more than 9 to 11 actors. For example, the Transfer, Append, PixPlot and Graph steps that are seen in the Pixie3D workflow in Appendix C, are clubbed together

into a single actor called the Diagnostics_3D actor workflow shown in the above figure. Similarly, the TransferArchiveText and the MakeMovieTemplate, in the above figure, are composed of the steps for the text archival and movie making in the original workflow, keeping the high level definition simple and not turning it into a complex graph with too many actors. The generated workflow follows the same set and number of steps but looks different from the original workflow giving a simpler view and structure.

Table 2 compares XGC workflow constructed manually and the generated by the wizard[#]. Considering the actors in the higher-level of the workflow only, it could be observed that there is a drastic decrease in the number of actors of the manually constructed workflow to the auto-generated one. However, there is a big difference in the ratio of the actors to the ones that are composite actors. But it does not make any difference to the overall execution performance of the workflow and rather keeps the higher-level view of the workflow simpler. The study revealed that the average number of underlying actors in a composite workflow is 3 and it is same in the auto-generated workflows also. However, we cannot compare that between the manually constructed workflows to the auto-generated one because the auto-generated workflow would always lag by one as compared to the manually constructed one. That is, if both the workflows were translated into graphs, then the depth of the auto-generated workflow would always be higher than the manually generated workflow by one. The number of parameters in either case remains the same and it does not affect the performance in any way.

[#] The visualization pipeline in the XGC workflow has not been considered. Only the initialization section and the diagnostic pipelines have been generated along with the archival components and have been compared to respective components of the original workflow.

Table 2 – Comparison between manual generated and automatically generated XGC Workflow

	Monitoring Workflows constructed manually	Monitoring Workflows constructing using the generator wizard
Number of actors at the higher level	27	12
Number of composite actors	9	10
Avg. Number of actors in each of these composite actors	3	3 (cant compare!)
Number of Parameters	remains the same	remains the same

The user does not have to worry about constructing the underlying sub-workflows. It is taken care of by the wizard. One of the other advantages of the wizard is that it reduces the overhead of parameterization by eliminating the need for repeatedly assigning values to the parameters. The most important reason for generating similar workflows automatically is to save the time. A workflow that is to be constructed manually could take days together while the wizard would construct the same workflow almost instantly. However, the additional components that may not have been implemented by the wizard have to be done manually, but it is still better to add the parts manually rather than building the whole workflow from the scratch.

To make it easier to test the generated workflows, we emulate the MySQL tables and the template library through Java. Doing so saves the trouble of updating the MySQL tables with different parameters that are long and complicated to type in every time a scenario needs to be tested. The generator creates an XML file by assembling the templates intelligently based

on the parameters from the user. Since the generator simply outputs a file, the overall time taken to generate the file is significantly small. The character-based reader/writer I/O classes of Java make the file writes efficient making the xml generation faster.

The manual development of the Pixie3D workflow was done based on the existing GEM workflow, because of which additional care had to be taken in order to ensure that the parameters are bound appropriately without any parameter holding a value that was valid in the GEM workflow but invalid in case of Pixie3D. In case of the wizard, the user has to give in the parameter values that are required by the workflow, while the generator takes care of binding the parameters with the right value.

Similarly we generated the GEM and the XGC workflows also using our wizard. We compared the execution of the workflows and observed the same behavior in the output logs that are produced by executing the auto-generated workflows as compared to the output logs produced by executing the original workflows. However, the auto-generated workflows are yet to be tested in the production environment. The following figures are the snapshots of the auto-generated XGC and GEM workflows.

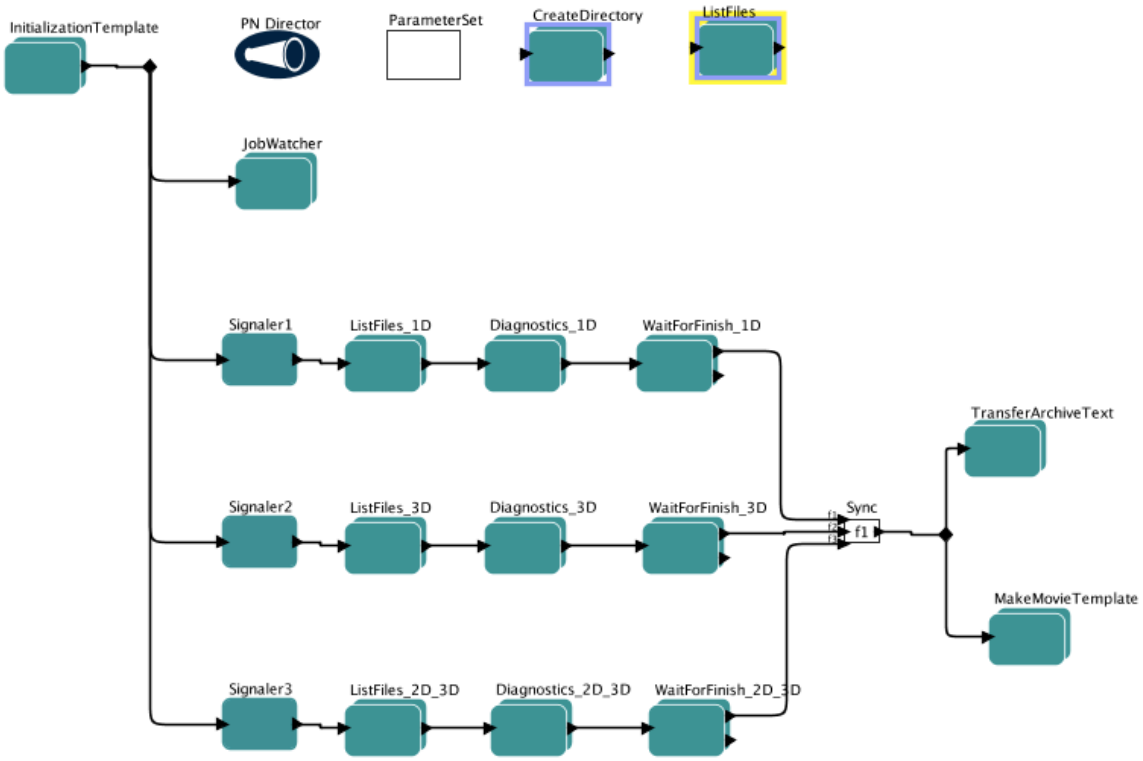


Figure 25 - Auto-Generated XGC Workflow

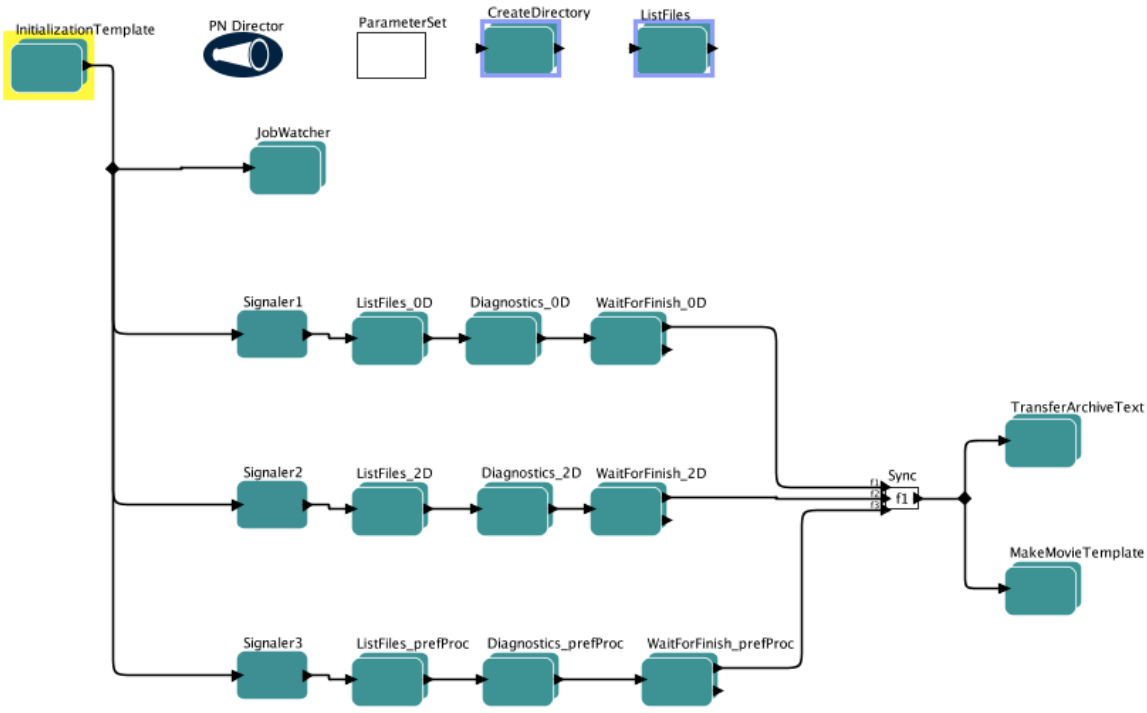


Figure 26 - Auto-Generated GEM Workflow

6.3.4 Availability of the code

The Java source code of the wizard implementation can be found http://drop.io/dhotqxx/asset/src-zip. The code generates a monitoring workflow with the diagnostic pipelines, performance monitoring pipeline, text archival, image archival and movie making components. The workflows generated by the wizard are based on the research conducted on the monitoring workflows created by ORNL. However, more modules can be created based on the additional options that the user needs to have. The workflow generator wizard is implemented in a modular way, hence every module is independent in nature and the integration of the module would just mean, adding the module method in one of the helper classes and referencing it in the generator class.

Chapter 7 - Conclusion and Future Work

Automation has long been a goal for scientists hoping to understand and implement information systems effectively. This thesis examines the challenge of developing reusable modeling components for scientific workflow generation. It proposes and provides a proof-of-concept implementation of the design for a workflow wizard tool that generates Kepler-based scientific workflows automatically by integration of some basic workflow components.

The work provides a compilation of analyses done on different monitoring workflows. Several design patterns and templates have been identified in the context of a family of workflows used by DOE. The application of these templates through the implementation of the *Workflow Generator Wizard* has been demonstrated. The thesis describes the synthesis of SDM's scientific monitoring workflows and provides a proof of concept by generating an existing workflow and comparing it with the original one using the SPA's Kepler workflow engine.

There are improvements to the implementation that need to be done like building the web pages (GUI), integrating the library with the Kepler library and using the *web_form* and *actor_process_file* at the backend. Besides that, it would be beneficial to store the auto-generated template in the library. The issue of eliminating the need for the *actor_port* table is open. A recommended way to eliminate it would be to make all the port names of every actor uniform. The Kepler uses the default location co-ordinates for every actor and therefore, on displaying the generated XML file in Kepler it is observed that all the actors get accumulated at the same location. In order to resolve the issue temporarily, our design creates the outer container dynamically while storing the inner contents as a template, leaving a room for the generator to produce co-ordinates dynamically and create a specification for location in the entity definition.

Another way of resolving the overlapping actors issue is to hard code the location property tags in the template library, which is not a good design although it does not affect the look or feel of the workflow. An aspect not included in this thesis is plotting the data for visualization using tools like AVS Express [AVS10] or VisIT (Visualization Tool) [Viz10]. The XGC workflow shown in Appendix A has a visualization pipeline besides the ones that are seen. While the implementation does not take care of the visualization part, it can be postulated as one of the options given to the user. The generator would then have a method that creates the visualization steps. The only changes to be made are adding an entry to the table and call the method at the right place.

The goal of this thesis was to experiment and analyze the possibility of generating a scientific workflow automatically by *a click of a button* given all the necessary parameters. This thesis provides the proof-of-concept of such system at a very basic level. However, for the future, the goal for the expansion of this concept should be a fully functional *Workflow Generator Wizard* that can build all types scientific workflows and not just the monitoring workflows considered in this thesis, thereby expanding the template repository to include many reusable components.

References

[Aals03] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. *Workflow patterns. Distributed Parallel Databases*, 14(1):5–51, 2003.

[Alti03] I. Altintas, S. Bhagwanani, D. Buttler, S. Chandra, Z. Cheng, M. A. Coleman, T. Critchlow, A. Gupta, W. Han, L. Liu, B. Ludascher, C. Pu, R. Moore, A. Shoshani, and M. Vouk. *A modeling and execution environment for distributed scientific workflows*. In *15th International Conference on Scientific and Statistical Database Management*, pages 247–251, July 2003.

[Alti04] I. Altintas, C. Berkley, E. Jaegar, M. Jones, B. Ludascher, and S. Mock. *Kepler: An extensible system for design and execution of scientific workflows*. In *SSDBM*, pages 21–23, 2004.

[Alti06] I. Altintas, B. Oscar, and E. Jaegar. *Provenance collection support in the kepler scientific workflow system*. In *Provenance and Annotation of Data*, volume 4145/2006, pages 118–132. Springer Berlin/Heidelberg, 2006.

[Alti07] I. Altintas, G. Chin, D. Crawl, T. Critchlow, D. Koop, J. Ligon, B. Ludäscher, P. Mouallem, M. Nagappan, N. Podhorski, C. Silva, and M. Vouk. *Provenance in kepler-based scientific workflow systems*. Poster in Microsoft eScience Workshop, 2007.

[Alti08] I. Altintas. *Lifecycle of scientific workflows and their provenance: A usage perspective*. In *IEEE Congress on Services-Part I*, pages 474–475. IEEE, 2008.

[AVS10] Advanced Visual Systems, Avs/Express High Performance Analysis (http://www.avs.com/software/soft_t/avsxps.html). Last accessed: 20-01-2010.

[Bark08] A. Barker and J. van Hemert. *Scientific workflow: A survey and research directions*. In R. Wyrzykowski and et al., editors, *Seventh International Conference on Parallel Processing and Applied Mathematics, Revised Selected Papers*, volume 4967 of *LNCS*, pages 746–753. Springer, 2008.

[Barr07] R. Barreto, T. Critchlow, A. Khan, S. Klasky, L. Kora, J. Ligon, P. Mouallem, M. Nagappan, N. Podhorski, and M. Vouk. *Managing and monitoring scientific workflows through dashboards*, October 2007.

[Barr94] R. Barret, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Vorst. *Templates - for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Siam, 1994.

[Bhar08] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi. *Characterization of scientific workflows*. In *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop*, pages 1–10, 2008.

[Bowe05] S. Bowers and B. Ludascher. *Actor-oriented design of scientific workflows*. In *24st Intl. Conference on Conceptual Modeling*. Springer, 2005.

[Bowe06] B. Ludascher, S. Bowers, T. Mcphillips, and N. Podhorszki. *Scientific workflows: More e-science mileage from cyberinfrastructure*. In *e-Science and Grid Computing, 2006. e-Science '06. Second IEEE International Conference on*, pages 145–145, December 2006.

[Bowe08] S. Bowers, T. M. McPhillips, S. Riddle, M. K. Anand, and B. Ludäscher. Kepler/ppod: *Scientific workflow and provenance support for assembling the tree of life*. In *Provenance and Annotation of Data and Processes*, volume 5272/2008, pages 70–77. Springer Berlin/Heidelberg, November 2008.

[BPEL07] IBM, BEA-Systems, Microsoft, SAP-AG, and Siebel-Systems. Business process execution language for web services version 1.1 (http://www.ibm.com/developerworks/library/specification/ws-bpel/). Last accessed: 08-02-2007.

[Brah07] S. Brahe and K. Schmidt. *The story of a working workflow management system*. In *GROUP '07: Proceedings of the 2007 international ACM conference on Supporting group work*, pages 249–258, New York, NY, USA, 2007. Sanibel Istand, Florida, USA, ACM.

[Buck02] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: a framework for simulating and prototyping heterogeneous systems. *Readings in hardware/software co-design*, pages 527–543, 2002.

[Chac04] L. Chacon. *Pixie3d: A parallel, implicit, extended mhd 3d code*. CEMM Meeting, November 2004.

[Chin10] G. Chin, C. Sivaramakrishnan, T. Critchlow, K. Schuchardt, and A. H. Ngu. Scientist-centered workflow abstractions via generic actors, workflow templates, and context-awareness for groundwater modeling and analysis. Submitted to the IEEE e-Science 2010 Conference, December 2010.

[Craw08] D. Crawl and I. Altintas. *A provenance-based fault tolerance mechanism for scientific workflows*. In *Provenance and Annotation of Data and Processes*, volume 5272/2008, pages 152–159, November 2008.

[Crit10] T. Critchlow. Scientific process automation improves data interaction (http://www.scientificcomputing.com/article-hpc-Scientific-Process-Automation-Improves-Data-Interaction-082809.aspx). Last accessed: 21-11-2010.

[Dani97] F. Daniels, K. Kim, and M. Vouk. *The reliable hybrid pattern a generalized software fault tolerant design pattern*. In *The 4th Pattern Languages of Programming Conference (PLoP'97)*, Washington University Technical Report, pages 97–34, Monticello, IL, September 1997.

[Davi07] S. Davidson, S. C. Boulakia, A. Eyal, B. Ludäscher, T. M. McPhillips, S. Bowers, M. K. Anand, and J. Freire. *Provenance in scientific workflow systems*. In *IEEE Data Eng*., volume 30, pages 44–50, December 2007.

[Davi08] S. Davidson and J. Freire. *Provenance and scientific workflows: Challenges and opportunities*. In *InProceedings of ACM SIGMOD*, pages 1345–1350, 2008.

[Davi99] J. I. Davis, M. Goel, C. Hylands, B. Kienhuis, E. A. Lee, X. Liu, J. Liu, L. Muliadi, S. Neuendorffer, J. Reekie, N. Smyth, J. Tsay, and Y. Xiong. *Ptolemy II: Heterogeneous concurrent modeling and design in java*. 1-3(UCB/ERL M05/21), July 2005.

[Deel05] E. Deelman, M. Livny, G. Mehta, A. Pavlo, G. Singh, M.-H. Su, K. Vahi, and R. Wenger. *Pegasus and DAGMan From Concept to Execution: Mapping Scientific Workflows onto Today's Cyberinfrastructure*, volume 16 of *Advances in Parallel Computing*. IOS Press, Amsterdam, The Netherlands, The Netherlands, March 2008.

[Deel08] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. *Pegasus: A framework for mapping complex scientific workflows onto distributed systems*. volume 13, pages 219–237, Amsterdam, The Netherlands, The Netherlands, 2005. IOS Press.

[Gamm95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Weasley, 1995.

[Gome08] C. Gomes, O. F. Rana, and J. Cunha. *Extending grid-based workflow tools with patterns/operators*. volume 22, pages 301–318, Thousand Oaks, CA, USA, 2008. Sage Publications, Inc.

[HDF510] The HDF Group, HDF5 (http://www.hdfgroup.org/HDF5/). Last accessed: 03-11-2010.

[Huhn94] M. P. Singh and M. N. Huhns. Automating workflows for service order processing: Integrating ai and database technologies. *IEEE Expert: Intelligent Systems and Their Applications*, 9(5):19–23, 1994.

[IBM10] IBM. New to soa and web services. http://www.ibm.com/developerworks/webservices/newto/index.html. Last accessed 21-11-2010.

[IBM010] IBM Web Services (http://www.ibm.com/developerworks/webservices/newto/websvc.html) Last accessed 18-08-2010.

[Kepler02] The Kepler User Manual (https://kepler-project.org/users/documentation).

[Kepler04] The Kepler Project (http://www.er.doe.gov/ascr/ProgramDocuments/ProgDocs.html), 2010

[Klas06] S. Klasky, B. Ludascher, and M. Parashar. *The center for plasma edge simulation workflow requirements*. In *ICDEW '06: Proceedings of the 22nd International Conference on Data Engineering Workshops*, page 73, Washington, DC, USA, 2006. IEEE Computer Society.

[Ku06] S. Ku, C. Chang, M. Adams, and et. al. Gyrokinetic particle simulation of neoclassical transport in the pedestal/scrape-off region of a tokamak plasma. In *Institute of physics Publishing Journal of Physics*, 46, pages 87–91, 2006.

[Lee00] E. A. Lee and S. Neuendorffer. *MoML — a modeling markup language in xml —* version 0.4. Technical Memorandum ERL/UCB M 00/12, University of California at Berkeley, 2000.

[Lee07] W. Lee. Third year status on scidac center for gyrokinetic particle simulation of turbulence transport in burning plasmas. Technical report, Princeton Plasma Physics Laboratory, 2007.

[Lee89] E. A. Lee, E. Goei, H. Heine, W. Ho, S. Bhattacharyya, J. Bier, and E. Guntvedt. *Gabriel: A design environment for programmable dsps*. In *DAC '89: Proceedings of the 26th ACM/IEEE Design Automation Conference*, pages 141–146, New York, NY, USA, 1989. ACM.

[Lin09] C. Lin, S. Lu, X. Fei, A. Chebotko, D. Pai, Z. Lai, F. Fotouhi, and J. Hua. *A reference architecture for scientific workflow management systems and the view SOA solution. IEEE Transactions on Services Computing*, 2:79–92, 2009.

[Luda06] B. Ludascher, I. Altintas, C. Berkley, D. Higgins, E. Jaegar, J. Matthew, E. A. Lee, J. Tao, and Y. Zhao. *Scientific workflow management and the kepler system*. volume 18, pages 1039 – 1065, Chichester, UK, 2006. John Wiley & Sons, Ltd.

[Luda09] B. Ludäscher, M. Weske, T. M. McPhillips, and S. Bowers. *Scientific workflows: Business as usual?* In *BPM '09: Proceedings of the 7th International Conference on Business Process Management*, pages 31–47, Berlin, Heidelberg, 2009. Springer-Verlag.

[Moua05] P. Mouallem. *Fault tolerance and reliability in scientific workflows*. Master's thesis, North Carolina State University, 2005.

[Moua09] P. Mouallem, R. Barreto, S. Klasky, N. Podhorski, and M. Vouk. *Tracking files in the kepler provenance framework*. In *Scientific and Statistical Database Management*, volume 5566/2009, pages 273–282. Springer Berlin/Heidelberg, 2009.

[Moua10] P. Mouallem, D. Crawl, I. Altintas, M. A. Vouk, and U. Yildiz. *A fault-tolerance architecture for kepler-based distributed scientific workflows*. In *SSDBM*, pages 452–460, 2010.

[OASI07] C. Barreto, V. Bullard, T. Erl, J. Evdemon, D. Jordan, K. Kand, D. Konig, S. Moser, R. Stout, R. Ten-Hove, I. Trickovic, D. vander Rijn, and A. Yiu. *The oasis committee: Web services business process execution language (ws- bpel) version 2.0* (2007).

[Podh07] N. Podhorski, B. Ludäscher, and S. Klasky. Workflow automation for processing plasma fusion simulation data. In *WORKS '07: Proceedings of the 2nd workshop on Workflows in support of large-scale science*, pages 35–44, Monterey, California, USA, 2007. ACM.

[Ptolemy10] The Ptolemy II Project (http://ptolemy.eecs.berkeley.edu/index.htm). Last accessed: 28-Oct-2010.

[Ptolemy03] E. A. Lee. Overview of the ptolemy project. Technical Memorandum UCB/ERL M03/25, University of California, Berkeley, University of California, Berkeley, CA, 94720, USA, July 2003.

[QiXu08] Q. Yu, X. Liu, A. Bouguettaya, and B. Medjahed. Deploying and managing web services: issues, solutions, and directions. *The VLDB Journal*, 17(3):537–572, 2008.

[Russ05] N. Russell, A. H. M. Hofstede, D. Edmond, and W. M. P. Van Der Aalst. Workflow data patterns: Identification, representation and tool support. *Lecture Notes in Computer Science, Conceptual Modeling - ER 2005*, 3716/2005:353–368, November 2005.

[SciDAC10] Scientific Discovery through Advanced Computing. SciDAC (http://www.scidac.gov/), Last accessed: 23-Jul-2010

[SciDAC08] From Data to Discovery (http://www.scidacreview.org/0602/html/data.html). The U.S. Department of Energy Labs. Last accessed: 22-May-2008

[SDMC09] Scientific Discovery through Advanced Computing (https://sdm.lbl.gov/sdmcenter/). The U.S. Department of Energy labs. Last accessed: 25-Jun-2009

[Shos09] B. Ludascher, I. Altintas, S. Bowers, J. Cummings, T. Critchlow, E. Deelman, D. De Roure, J. Freire, C. Goble, M. Jones, S. Klasky, T. M. McPhillips, N. Podhorski, C. Silva, I. Taylor, and M. Vouk. *Scientific process automation and workflow management*. In A. Shoshani and D. Rotem, editors, *Scientific Data Management*, Computational Science Series, book 13. Chapman & Hall, 2009.

[Shos10] A. Shoshani. Sdm center semi annual report for 2010. 2010.

[Sing05] M. Singh and M. Huhns. *SERVICE-ORIENTED COMPUTING Semantics, Processes, Agents*. John Wiley & Sons, Ltd., 2005.

[Sing94] M. P. Singh and C. Tomlinson. Workflow execution through distributed events. In *COMAD*, 1994.

[Sing96] Singh M.P., Vouk M.A., "*Scientific workflows: scientific computing meets transactional workflow*s," Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems: State-ofthe-Art and Future Directions, Univ. Georgia, Athens, GA, USA; 1996, pp.SUPL28-34

[SOAP00] D. Box and et. al. Simple object acess protocol (soap) 1.1 (http://www.w3.org/TR/2000/NOTE-SOAP-20000508/). Last accessed: 08-May-2000.

[Stoi06] K. P. Stoilova and T. A. Stoilov. Evolution of the workflow management systems. In *XLI International Scientific Conference on Information, Communication and Energy Systems and Technologies - ICEST*, pages 225–228, Sofia, Bulgaria, June 2006.

[Tan09] W. Tan, P. Missier, R. Madduri, and I. Foster. Building scientific workflow with taverna and bpel: A comparative study in cagrid. In *Service-Oriented Computing — ICSOC 2008 Workshops: ICSOC 2008 International Workshops*, pages 118–129, Berlin, Heidelberg, 2009. Springer-Verlag.

[Tayl07] I. Taylor, M. Shields, and A. Harrison. *The triana workflow environment: Architecture and applications*. In I. Taylor, E. Deelman, D. Gannon, and M. Shields, editors, *Workflows for e-Science*, pages 320–339, 2007.

[Tcho10] R. Tchoua, S. Klasky, N. Podhorszki, B. Grimm, A. Khan, E. Santos, C. Silva, P. Mouallem, and M. Vouk. *Collaborative monitoring and analysis for simulation scientists*. In *CTS 2010*, 2010.

[UDDI10] OASIS. uddi xml.org. Last accessed: 23-Jul-2010.

[USDOE10] U.S DOE Program Document Archive (http://www.er.doe.gov/ascr/ProgramDocuments/ProgDocs.html) Last accessed: 16-Nov-2010.

[Viz10] The Visit Software, LLNL (https://wci.llnl.gov/codes/visit/home.html). Last accessed: 23-Jun-2010.

[Vouk07] M. A. Vouk, I. Altintas R. Barreto, J. Blondin, Z.Cheng, T. Critchlow, A. Khan, S. Klasky, J. Ligon, B. Ludaescher, P. A. Mouallem, S. Parker, N. Podhorszki, A. Shoshani, C. Silva" "*Automation of Network-Based Scientific Workflows,"* IFIP, Vol 239, "*Grid-Based Problem Solving Environments, eds. Gaffney PW and Pool JCT*" (Boston: Springer), pp. 35-61, 2007.

[Vouk09] Mladen Vouk, "*Cloud Computing – Issues, Research and Implementations*," Journal of Computing and Information Technology, Vol 16 (4), 2008, pp 235-246

[Vouk97] M. A. Vouk and M. P. Singh. *Quality of service and scientific workflows*. In *Proceedings of the IFIP TC2/WG2.5 working conference on Quality of numerical software*, pages 77–89, London, UK, UK, 1997. Chapman & Hall

[WMC10] Workflow Management Coalition (http://www.wfmc.org/). Last accessed: Nov-2010.

[WMC95] D. Hollingsworth. Workflow management coalition, the workflow reference model. Workflow Management Coaalition Specification TC00-1003, Workflow Management Coalition, Hampshire, UK, January 1995.

[WSDL01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description

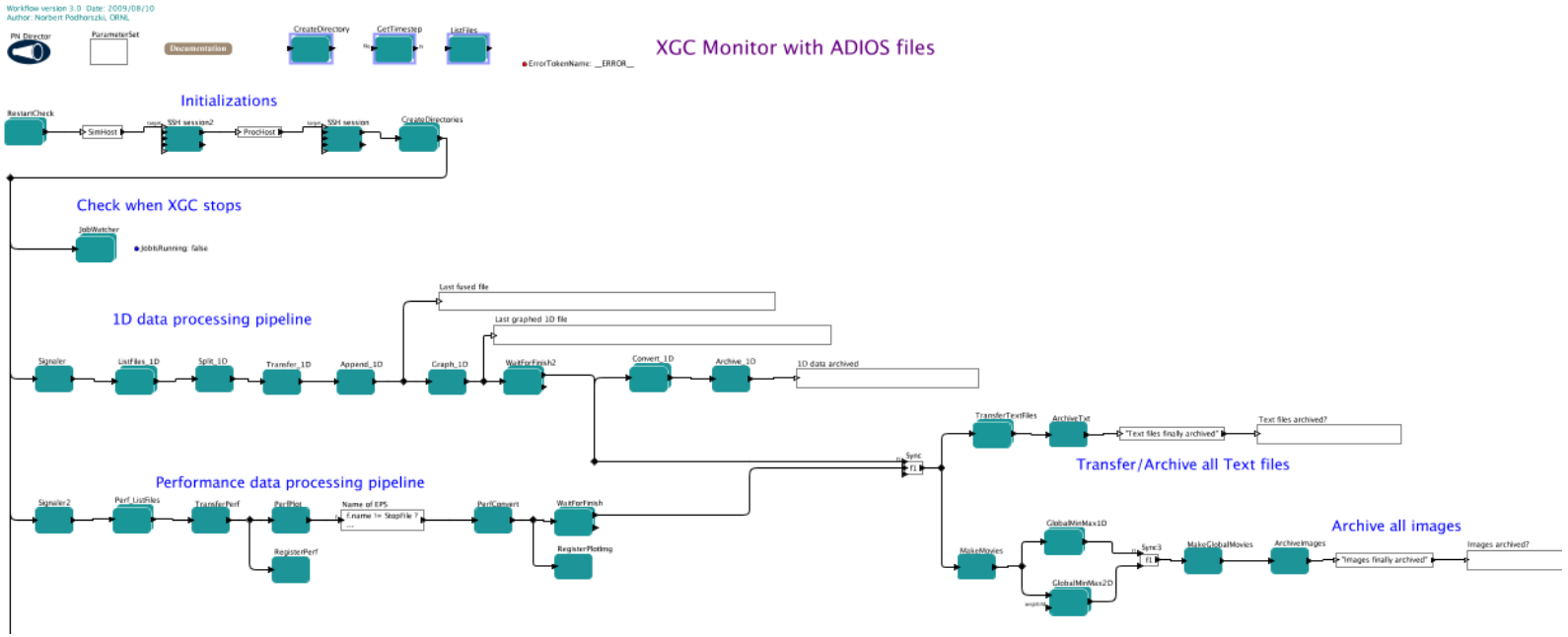
language (wsdl) 1.1 (http://www.w3.org/TR/wsdl). Last modified: 15-Mar-2001.

[WWF01] Microsft. Windows workflow foundation (http://msdn.microsoft.com/en-us/library/ms735967%28VS.90%29.aspx). Last Published: 03-Mar-2010.

[Yild09] U. Yildiz, A. Guabtni, and A. H. Ngu. *Towards scientific workflow patterns*. In WORKS '09: Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science, New York, NY, USA, 2009. ACM.

Appendices

Appendix A – XGC



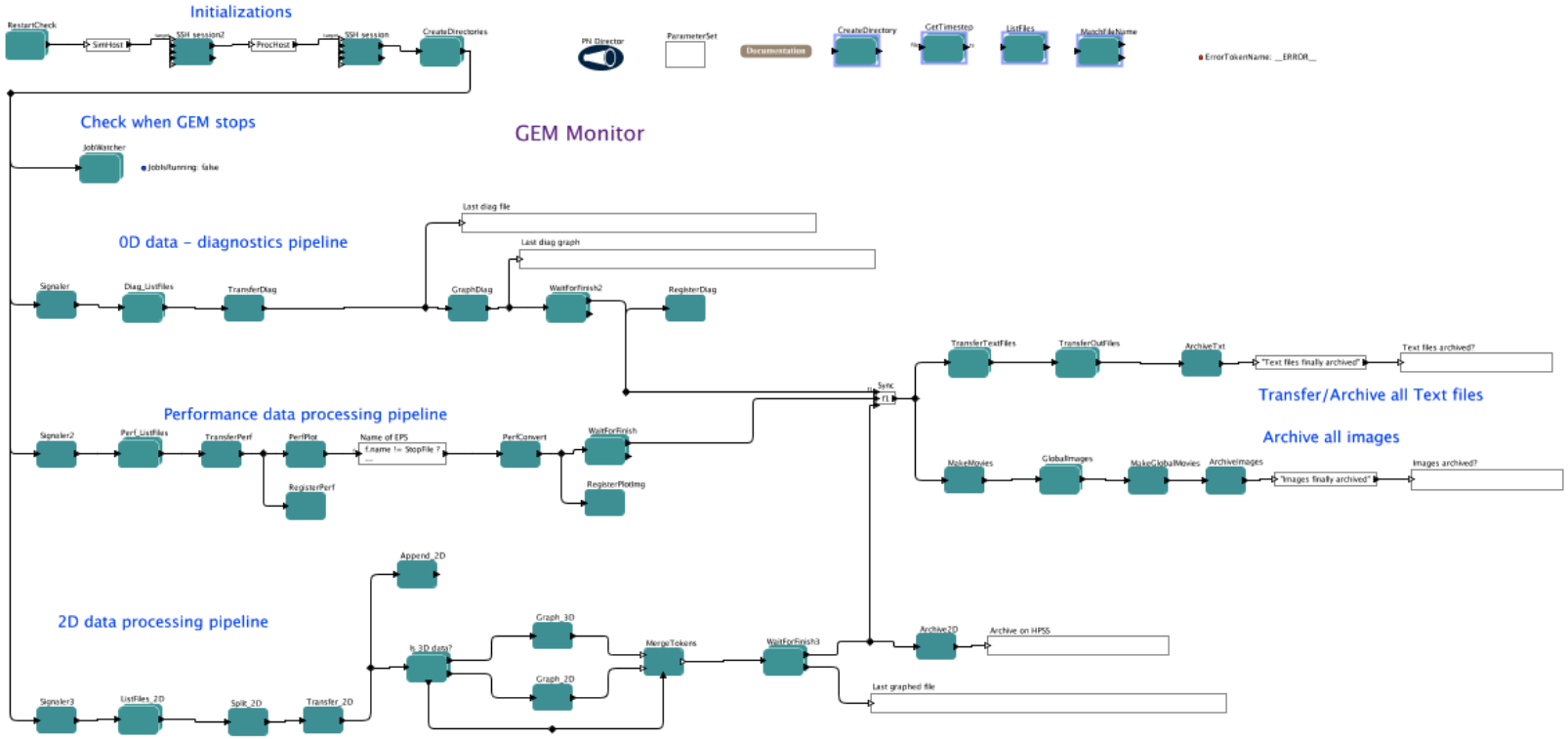
XGC Monitoring Workflow [6]

Author – Norbert Podhorski, ORNL

[6] Although the snapshot shown above does not illustrate, the XGC workflow has one more pipeline for the visualization called AVS express.

The Center for Plasma Edge Simulation Project (CPES) is a part of the Fusion Simulation Projects funded by the DOE. CPES has created an integrated predictive plasma edge simulation code package to understand localized mode instabilities, which reduce the confinement properties of the plasma and shorten the wall's lifetime [Podh07]. They have developed new models for the plasma edge in a kinetic regime with complex geometry [Klas06]. XGC [Ku06] consists of a set of two different particle simulation codes (XGC-0 and XGC-1), produced as a part of the CPES project. XGC-1 is a full-f gyrokinetic ion-electron particle code derived from XGC-0, designed for integrated simulation of neoclassical and electrostatic turbulence physics using enormous parallel processors.

Appendix B – GEM

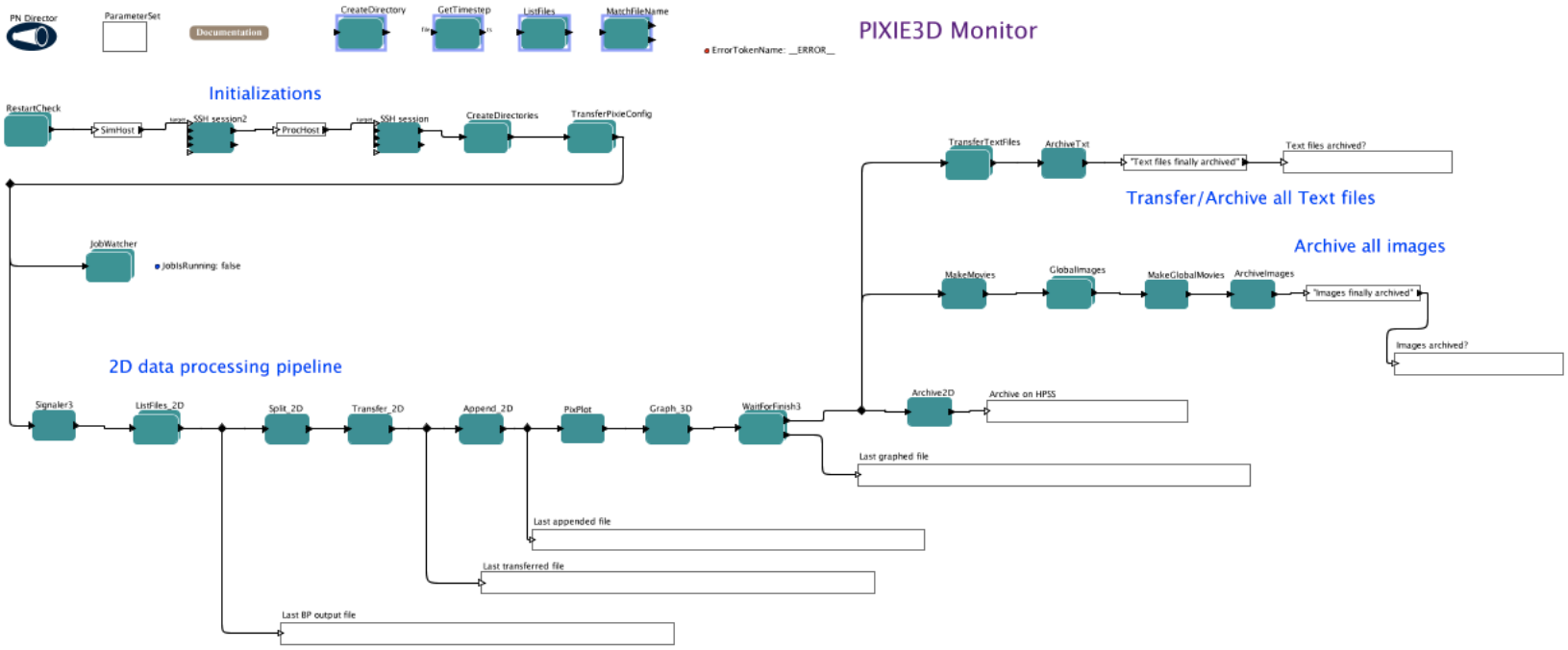


GEM Monitoring Workflow

Author – Norbert Podhorski, ORNL

Gyrokinetic Electromagnetic Turbulence Simulation (GEM) is one of the simulations conducted by the SciDAC center of gyrokinetic particle simulation of turbulent transport in burning plasmas [Lee07]. It is a delta-f particle code consisting of the dynamics of the gyrokinetic ions and drift-kinetic electrons. The code accurately measures the magnetic fluctuations by studying the well-magnetized plasma physics. It is a powerful code, which can measure up to extremely low fluctuation levels.

Appendix C – PIXIE 3D

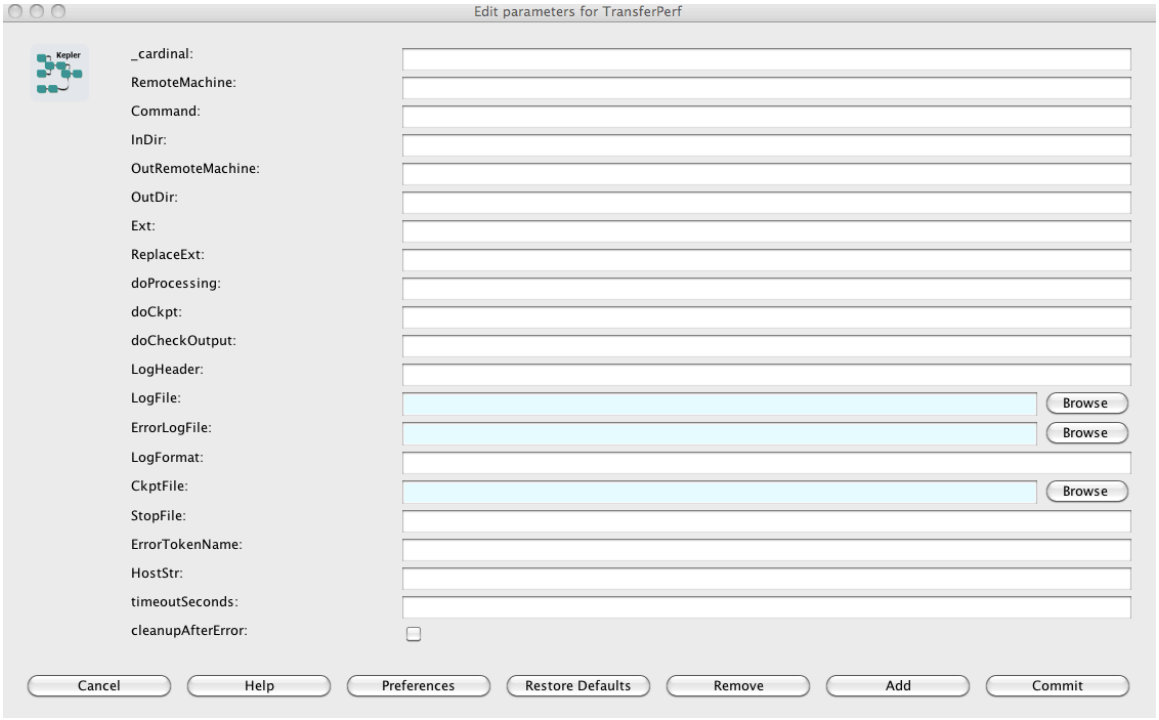


Pixie3D Monitoring Workflow

Authors – Harini Iyer, NCSU, Norbert Podhorski, ORNL

Parallel, Implicit, eXtended MHD 3D Code (Pixie3D) [Chac04] is a parallel and fully implicit 3D extended MHD solver. The goal of the Pixie3D project is to demonstrate the path for fully implicit MHD in general geometries, using state of art scalable solver technology and exploiting massively parallel computing environments. Appendix C contains the snapshot of the Pixie3D workflow.

Appendix D – ProcessFile Actor Parameters



Parameter Set for ProcessFile actor

Appendix E – VisIT interface

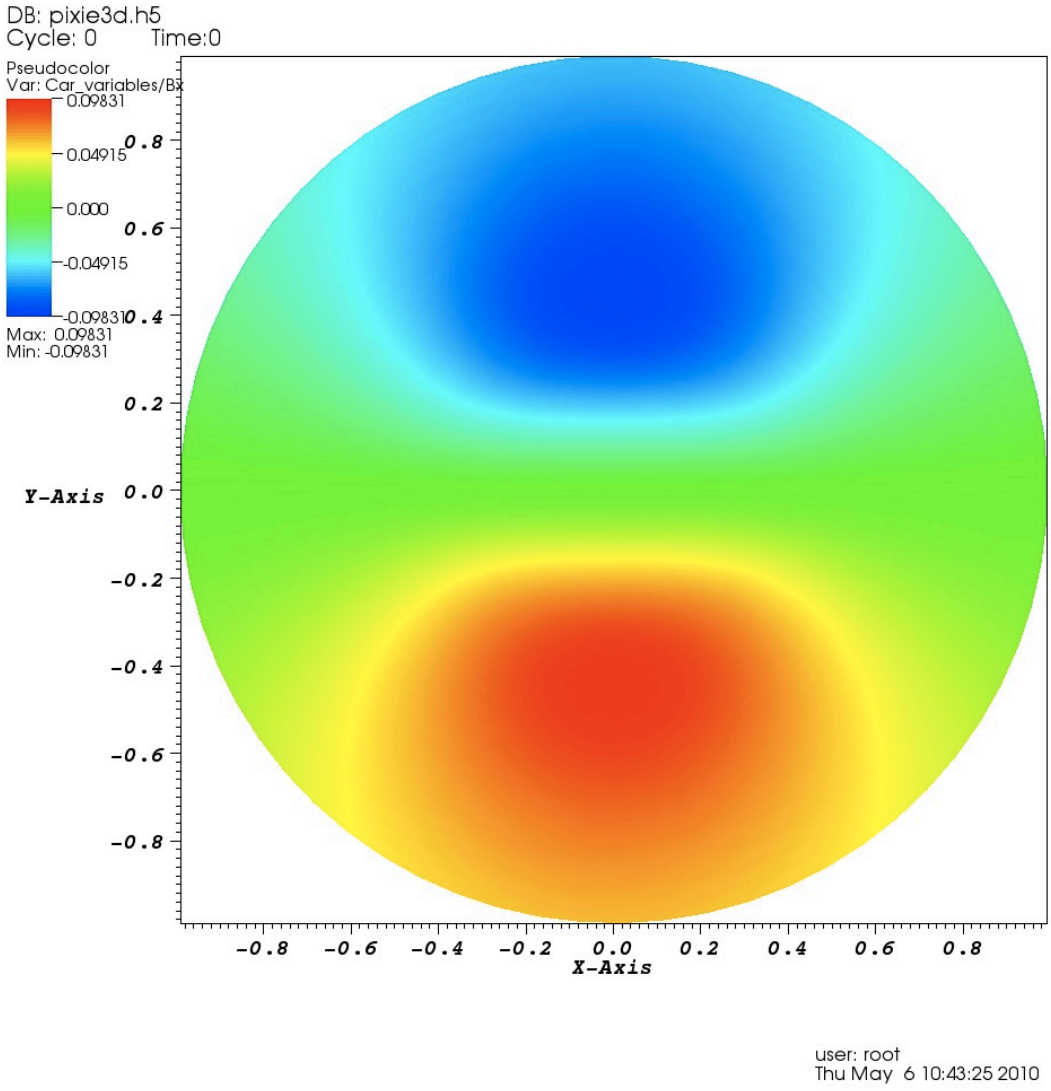


Image plotted from the Pixie3D data using VisIT command-line client-server model

Appendix F: Acronyms and Defintions

BPEL – Business Process Execution Language

DMA - Data Mining and Analysis

GEM – Gyrokinetic Electromagnetic Turbulence Simulation

HPSS – High Performance Storage System

KPR – Kepler Provenance Recorder

Pixie3D - Parallel, Implicit, eXtended MHD 3D Code

PR – Provenance Recorder

ORNL – Oak Ridge National Laboratory

SciDAC - Scientific Discovery through Advanced Computing

SDM – Scientific Data Management

SEA - Storage Efficient Access

SOA – Service Oriented Architecture

SPA – Scientific Process Automation

SWfMS - Scientific Workflow Management Systems