European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
A Network of Excellence funded by the European Commission

Dynamicity in Scientific Workflows

Manuel Caeiro-Rodríguez
mcaeiro@det.uvigo.es
University of Vigo
ETSI Telecomunicacin, Campus Universitario s/n,
E-36200 Vigo, Spain

Thierry Priol
Thierry.Priol@inria.fr
INRIA, Campus Universitaire de Beaulieu,
35042 Rennes, Cedex

Zsolt Németh
zsnemeth@sztaki.hu
MTA SZTAKI Computer and Automation Research Institute
P.O. Box 63, Budapest, H-1518, Hungary

CoreGRID Technical Report
Number TR-0162
31 August 2008

Institute on Grid Information, Resource and Workflow Monitoring Services

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

Dynamicity in Scientific Workflows

Manuel Caeiro-Rodríguez
mcaeiro@det.uvigo.es
University of Vigo
ETSI Telecomunicacin, Campus Universitario s/n,
E-36200 Vigo, Spain

Thierry Priol
Thierry.Priol@inria.fr
INRIA, Campus Universitaire de Beaulieu,
35042 Rennes, Cedex

Zsolt Németh
zsnemeth@sztaki.hu
MTA SZTAKI Computer and Automation Research Institute
P.O. Box 63, Budapest, H-1518, Hungary

CoreGRID TR-0162
31 August 2008

Abstract

Dynamicity is a recurrent topic in traditional business workflow systems. The need and feasibility to perform changes in workflow process instances while they are being executed has been a main (and to a long extend yet unsolved) challenge. More recently, the scientific workflow domain has also paid attention to this topic and some of the current scientific workflow management systems give a certain support for dynamicity. In general, there is a common agreement that dynamicity is an intrinsic requirement for scientific workflows, but the understanding about the real needs and functionality to be provided is confuse. This report is mainly focused on contributing to enhance such an understanding by analyzing dynamicity scenarios, requirements and proposals in scientific workflows. First, five general scenarios involving different dynamicity needs are described through the introduction of concrete examples. Then, these scenarios are used to identify a set of dynamicity requirements for scientific workflows support. Finally, a review of current well-known scientific workflow execution systems is presented, focusing on their proposals to support dynamicity.

1 Introduction

The use of computational resources to perform scientific activities is increasing very fast in all scientific disciplines. Nowadays, numerous large-scale computational systems are used to support research in physics (e.g., the CERN Large Hadron Collider Computing Grid[1]), in biology (e.g., the National Center for Biotechnology Information[2]), in geology (e.g., Dissemination and Exploitation of GRids in Earth sciencE[3]), in oceanography (e.g., Laboratory for the Ocean

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

[1] http://lcg.web.cern.ch/LCG/
[2] http://www.ncbi.nlm.nih.gov/
[3] http://degree.ipgp.jussieu.fr/

Observatory Knowledge INtegration Grid [4]), in astronomy (e.g., the European Grid of Solar Observations[5]), etc. These systems are characterized by the distributed and heterogeneous nature of the involved resources and by the need for the effective management and operation of the large amount of resources, data and processes involved. Eventually, human scientists are the responsible of this management and operation. Nevertheless, in an ideal context, they should worry as less as possible about all these management and operation issues and focus their attention on the particular scientific concerns.

Several technologies have been proposed to support human scientists in the management and operation of large-scale computational resources. The *Grid* paradigm [16] is an abstraction to a large collection of distributed heterogeneous resources, including computational, storage and instrument elements, controlled and shared by different organizations. Initially, scientists' requirements have been to deploy applications over these Grid systems, often by manual selection and invocation of the resources to use. However, this is evolving into the need to deploy not single applications, but whole scientific processes with several interdependent applications. Now, it is needed to deploy the different applications, to control the order in which applications are performed, to manage the transfer of data among applications, etc. Moreover, these operations should be performed automatically, without the need for human intervention except where desired. At this point, scientific workflows have been proposed [85, 80]. Scientists are intended to specify scientific processes, including all the intended applications and dependencies. Then, a scientific workflow management system will execute (or enact) such specifications carrying out the intended processes while offering to the scientists an appropriate interface to monitor the progress, visualize the output and steer their execution.

A main requirement in scientific workflows is the support of changes during execution [7, 36, 47, 57]. On the one hand, the scientific context of the user is in flux as the scientific exploration process evolves and new evidences and ideas are considered [77]. On the other hand, the large-scale computational system where the workflows operate over is also in flux, as networks, platforms and other resources come and go. Therefore, scientific workflow management systems should support dynamic, adaptive and user-steered processes. Reproducibility is also an important requirement in order to enable the study of the experiments and their comparison.

A first approach to dyminacity in scientific workflows requires a precise definition of this concept[6]. There are different understandings and the various proposals approach its support focusing on the solution of different problems. Therefore, a main goal of this report is to provide an updated picture of this research field and to identify scientific scenarios where dynamicity is required. Then, further goals are the identification of dynamicity requirements and the description of existing techniques and proposals.

This report needs to be considered in the context of a more specific research. Our final goal is to explore the feasibility of a *Chemical Programming* (CP) [12] solution to support the dynamicity needs in scientific workflows. The CP paradigm, or more specifically the *Higher-Order Chemical Language* (HOCL) [11], offers a natural parallelization model of the computation that resembles quite well the distribution of processing chunks that is performed in Grids. In addition, it is considered that HOCL programs can perform well in the presence of changes, facilitating their modification during execution. These two properties seem a good starting point to support dynamicity in scientific workflows.

The remainder of this report is organized as follows. Firstly, a view of the scientific workflow domain and the main issues involved is introduced. Then, section 3 analyzes five general scientific scenarios demanding dynamicity. Next, section 4 identifies and analyzes the main requirements related with the support of dynamicity. Then, a section analyzes techniques and approaches to support some of such requirements. In the next section, some of the most popular scientific workflow management systems are reviewed focusing on the techniques and approaches identified. The report finishes analyzing the differences in dynamicity between scientific and business workflows and the main conclusions of this work.

2 Scientific Workflows

The term "workflow" has traditionally been used in the context of business workflows. The *Workflow Management Coalition* (WfMC[7]), a global organization focused on business workflow, defines a workflow as [43]: "*the computerized facilitation or automation of a business process, in whole or part*". In a similar way, a scientific workflow is

[4] http://lookingtosea.ucsd.edu/

[5] http://www.egso.org/

[6] See http://rubyhacker.com/coralbook/dynam.html for a discussion around the dynamicity meaning

[7] WfMC Web site http://www.wfmc.org/

viewed as a description of the processes that a scientist needs to execute in order to create a "scientific output" [54]. In an metaphorical way, a workflow can be viewed as a "cooking recipe". The recipe involves several tasks (e.g., to fry, to mixture) to be performed using certain tools (e.g., a pan, a spoon) and resources (e.g., olive oil, potatoes, eggs). In addition, sometimes, tasks may need resources that have to be produced by other tasks previously (e.g., before introducing a cake in the oven it is necessary to mixture all the ingredients). Other times, a certain synchronization among tasks is required (e.g., to prepare an Hollandaise sauce butter has to be added very slowly to a mixture of egg yolks and lemon juice, while the mixture is being continually beaten). Two key issues in this description are *data flow* (namely, transfer of "contents" between tasks) and *control flow* (namely, transfer of the "execution right" between tasks). In addition, each task requires some "active resources" (e.g., the chef) to be performed. It is quite obvious that these concepts can be applied in a scientific context. Many scientific applications can be arranged as "scientific recipes": tasks to be performed; data transfers between tasks; etc.

The main idea in the workflow approach is to separate the workflow process description or specification from the system responsible of its execution. The workflows are typically modeled, and graphically depicted, as "Nodes" (representing tasks) interconnected by directed "Links" (representing data or control flows/dependencies). Then, scientific workflow specifications can be executed (or enacted) by an appropriate scientific execution software, named as *Workflow Management System* (WfMS). In accordance with the WfMC a WfMS is defined as [43]: "*A system that completely defines, manages and executes workflows through the execution of software whose order or execution is driven by a computer represention of the workflow logic*".

A basic architecture of a typical scientific WfMS involving four main elements is described in [87] (cf. figure 1):

- A *Workflow* specification containing the process description ("the recipe") with tasks, flows and dependencies.
- The available computational *Resources*. In the domain of scientific workflows it is usually distinguished between services and resources.
- A *Engine* to orchestrate the system behavior. This component is responsible for interpreting the workflow description and "bring it into life" (also known as enactment).
- A *Middleware* that realizes the communication and binding among the *Resources* and the *Engine*. Particularly, it should support the transfer of data and the invocation of operations.

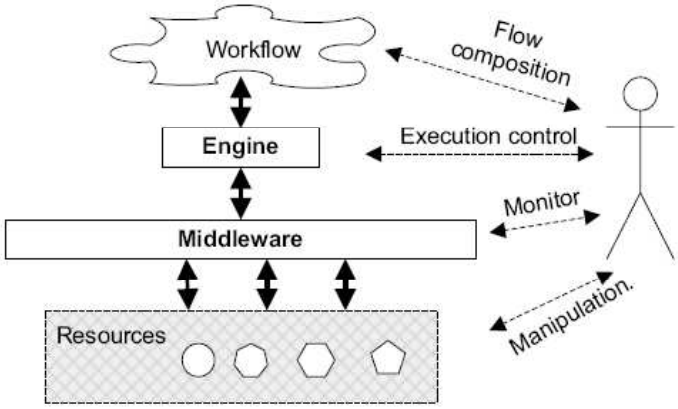

Figure 1: Main components of a Grid workflow system (image taken from [87])

This basic structure can involve several differences at certain issues between the existing scientific workflow management systems (see section 6). They use different names for the task and link concepts [78]. Tasks are named as *actors* in Kepler, *transitions* in Petri Net based proposal, *procedures* in VDL, *activities* in AGWL, *elements* in CoG Kit's Karajan, and *units* in Triana. The links between tasks are also known by different names: *vertices* in a graph, *edges* in Petri Nets, *pipes* in data-flow systems, and *channels* in service-based systems. In the case of the links the differences are not simply in the sintaxis, but also in their semantic meaning where it is possible to distinguish between

control- and data-driven models. Another difference between scientific workflow systems is in the final "actors" used to perform the tasks, distinguishing between resources and services. In the following sections these differences are analyzed together with the description of the common life-cycle of a scientific workflow. Eventually, some of these issues play an important role in the consideration of dynamicity needs.

2.1 Control- Versus Data-Driven Workflows

The meaning of links in scientific workflows can be considered in different ways. Two main approaches are followed, control-driven and data-driven, but intermediate approaches mixturing elements from both are also proposed [78].

In the *control-driven approach*, links between tasks represent control constrains to the performance of each task. Several control structures can be found, being the more basic the *sequence*, that represents ordered execution of tasks. More complex constructs are *splits*, *joins*, *loops*, etc. The workflow patterns have been proposed in the business workflow domain to catalog and describe the possible control behaviors [2].

In the *data-driven approach*, links between tasks represent data dependencies. A task consumes data and produce data. Each task can be initiated as soon as its input data is available. his approach has the advantage that parallel execution of independent tasks is modeled for free [46].

There are different arguments to use one or another approach. In the one hand, using the control driven approach gives more control over the actual order of task execution [68]. In the other hand, the data driven representations are simpler for the final user. Pure data-driven approach, however, is not expressive enough to model iterative behavior [68].

Many proposals follow an hybrid approach based on a combination of control and data approaches. These hybrids use both types of dependencies, but they are normally biased towards either data flow or control flow, using the other to better handling certain conditions. As an example, in JOpera [69] a data-driven solution is taken as base proposal and then particular control constructs are included for those patterns that are not well-supported. Well known scientific workflow languages, such as AGWL (see section 6.1) and SCULF ((see section 6.6) include control-flow (e.g., exclusive choice, sequential loops) and data-flow links (e.g., input/output ports, data collection operators [73]).

2.2 Workflow Processing Strategies: Resource-based and Service-based

In relation with the final actors responsible for the effective performance of tasks two main solutions are identified [37]:

- The *task-based* solution (also referred to as *global computing*), where a computing task is formally described before being submitted. Users define computing tasks to be executed. Executable code files, input data files and comand-line parameters are provided to enable the invocation of the execution. The workflow manager has to find an appropriate computational resource where the program can be executed.

- The *service-based* solution (also referred to as *meta computing*), where a computation handled by an external service is invoked through an interface. Services are seen as black boxes from the workflow manager, for which only the invocation interface is known. The workflow manager has to find appropriate services providing the functionality required.

These approaches have led to the design of different workflow managers. The services paradigm has been widely adopted by middleware developers for the high level of flexibility it offers. However, this approach is less common for application code, as it requires all codes to be instrumented with a service interface. In practice, there are intermediate solutions where services are dynamically deployed in an on demand basics.

2.3 The Scientific Workflow Life-cycle

The execution of scientific workflows involves the processing of a workflow specification and the coordinated use of appropriate resources or services. Several authors identify three stages in this process, proposing three different levels for workflow descriptions [30, 57]:

- *Abstract workflows.* At this high level of abstraction the workflow contains just information about what have to be done at each task along with information about how tasks are interconnected. There is no notion of how input

data is actually delivered or how tasks are implemented. As an example, an abstract workflow may be described as being a linear equation solver taking inputs from an experimental device and sending the results to a monitor.

- *Concrete workflows*. The mapping down to a concrete workflow annotates each of the tasks with information about the implementation and/or resources to be used. Information about method invocation and actual data exchange format are also defined. For example, a Cholesky solver will be used.
- *Workflow Instances*. The workflow instance represents the actual mapping of the concrete workflow onto the computational resources, involving the specific input data and the corresponding results. This level can be decomposed into two additional ones [27]:
  - *Workflow instance*. Describes the workflow at the application level without indicating the resources needed to execute it. Then, it involves the concrete workflow and the data, but not any details about the executing system.
  - *Executable workflow*. It is created by mapping the workflow instance onto the computing resources.

This notion of levels is not recognized by all the proposals and in practice is quite restrictive. Rarely, will a workflow pass cleanly through these three stages. Parts of the same workflow may remain abstract at the same time as other parts are concrete (or even enacted). Nevertheless, it can be important to have in mind these three stages in order to analyze different dynamicity needs.

3 Intrinsic Dynamicity in Scientific Workflows

This sections shows how dynamicity is intrinsic demanded by scientific workflows. Figure 2 represents the different stakeholders involved in the scientific workflow model. They are taken into account to identify five scenarios (A to E) where it is clear the need for dynamicity. The scenarios consider different issues, some of then more related to the scientific problems that need to be supported (scenarios A to C) and other ones more related to the computational mechanisms used to provide the support (scenarios D and E).

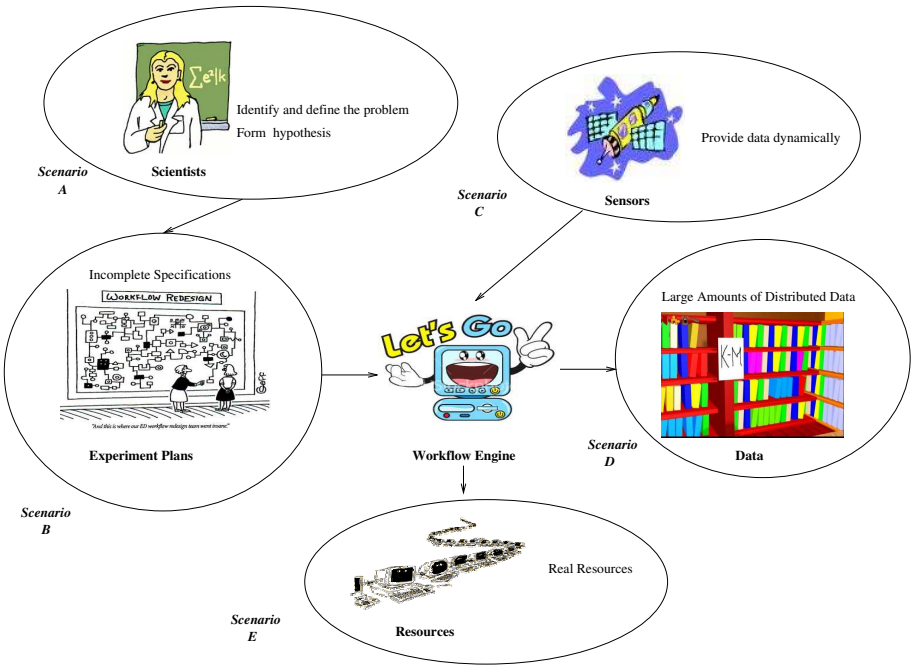

Figure 2: Scientific workflow scenarios requiring the support of dynamic behaviors

3.1 Scenario A: Scientific Exploration

Scientific research is exploratory in nature. The scientific method [1], the "tool" that scientists use to perform scientific research, is described as an iterative process involving the following steps: identifying the problem, stating a hypothesis, conducting experimentation, evaluating the results and reaching a conclusion. A key point in this process is the testing of hypothesis through experimentation and observation. Usually, this requires the performance of experiments involving different activities with several parameter settings. With the formulation of experiments as scientific workflows, one could imagine creating a first workflow description of an experiment and subsequently testing with different combinations of parameters and process adaptations until a suitable solution is found. In this way, scientists do not use to work with well-established workflows, rather they converge on final workflows through an iterative process of exploration, often in a trial and error manner, developing different versions of the same workflow as the research progresses [36].

A important domain where this kind of exploratory scenario can be found is *Medical Image Analysis* (MIA) [35]. MIA procedures use to involve the processing of medical image data in accordance with several elementary tasks depending on the analysis objectives. In general, the chain of elementary tasks to apply is not necessarily linear and there is often a graph of interconnected tasks. For efficiency in healthcare, well-established MIA methods are typically automated or require minimal user intervention [66]. Nevertheless, when existing procedures and techniques are not valid, it is necessary to conduct medical experiments resembling the scientific method [60, 17, 56, 79, 53, 76, 18]. In these situations, an expert needs to conceive several procedures over different data sets and to assess the computation results. By inspecting the results produced during experimentation, a medical or technical expert can detect problems and conceive new solutions.

A more concrete example showing this exploratory behavior in MIA is described in this paragraph. It is about the identification of brain tissue loses in order to diagnosticate *Multiple Sclerosis*. This degenerative disease is a neurological disorder that predominately affects young adults and is associated with recurrent attacks of focal inflammatory demyelination (plaques) that cause neurological impairment, separated by periods of relative stability. An automatic image-based method to quantify disease burden is described in [21], see figure 3. The fully automated method uses *Magnetic Resonances* (MR) to quantify brain atrophy and is based on estimation of the *Brain to IntraCranial Capacity Ratio* (BICCR). The stages involved in this procedure are basically:

1. *Intensity Non-Uniformity Correction*. The inhomogeneity of the MR acquisition device magnetic field introduces a bias perceptible in images as a continuous variation of gray-level intensity. The non-uniform intensity correction algorithm iteratively proceeds by computing the image histogram underestimating a smooth intensity mapping function that tends to sharpen peaks in the histogram. Application of this procedure improves the accuracy of the tissue classification stage described below.

2. *Stereotaxic Registration*. Each image is linearly registered in a common Talairach space in order to compensate for size variations between individuals. The registration algorithm proceeds with a coarse-to-fine approach by registering subsampled and blurred MR image volumes with the stereotaxic target. The final data used for subsequent processing is only re-sampled once to minimize re-sampling/interpolation artifacts.

3. *Intensity Normalization*. In preparation for intensity-based classification, each image is intensity normalized to an average PDW or T2W target volume already in stereotaxic space.

4. *Cropping*. Since the entire cerebrum was not covered by the MRI acquisition in all subjects, the inferior and superior slices were cropped away.

5. *Anisotropic Diffusion*. It has been shown that the application of an edge preserving noise filter can improve the accuracy and reliability of quantitative measurements obtained from MR images.

6. *Bayesian Classification*. A Bayesian classifier is then used to identify all gray-matter, white-matter, cerebrospinal fluid, lesion and background voxels.

7. *Brain Masking*. Mathematical morphology was used to eliminate the scalp and meninges for further processing. A brain mask was created by applying an opening operator.

This algorithm was achieved after a large research, following an iterative exploration on different stages and datasets. Actually, only the two last stages are directly related with the identification of the brain atrophy. The first

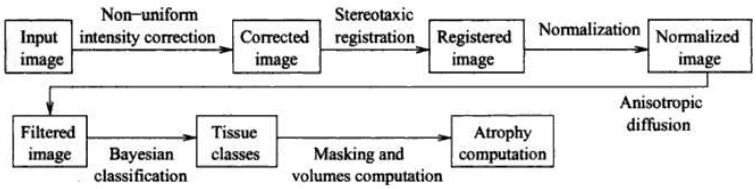

Figure 3: Diagram of the atrophy computation method stages for multiple sclerosis (image taken from [21])

stages involve basically the preparation of the images in order they can be correctly processed. There are three stages introduced to obtain an appropriate quality in the images: *Intensity Non-Uniformity Correction*, *Intensity Normalization* and Anisotropic Diffusion. Depending on the raw images features these stages may be required or not. Some of these are dependent on the hardware profiles of the data collection device used to generate the datasets. In addition, the *Stereotaxic Registration* and the *Cropping* are introduced in order to rectify discrepancies in the raw data. Since a patient may be scanned with different equipment over the course of a treatment, a registration stage may be needed to align each of the patient's scans.

An exploration process can involve the experimentation with different tasks, as in the previous example, but it can also be considered in relation with variations in the range of parameters used. The *Promoter Identification Workflow* (PIW) [7] is a well-described workflow to identify and characterize eukaryotic promoters (a promoter is a subsequence of a chromosome that sits close to a gene and regulates its activity). Starting from microarray data, cluster analysis algorithms are used to identify genes that share similar patterns of gene expression profiles that are then predicted to be co-regulated as part of an interactive biochemical pathway. Given the gene-ids, gene sequences are retrieved from a remote database and fed to a tool that finds similar sequences. In subsequent steps, transcription factor binding sites and promoters are identified to create a promoter model. An improved version of PIW allows the user to inspect intermediate results and select and re-rank them before feeding to subsequent steps, enabling the iterative and interactive refinement of the promoter model. Similar examples related with the exploration of parameters can be found in MIA [66] or in biology [45].

This scenario is being considered by numerous scientific workflow initiatives as it can be found in many projects: gene expression analysis [45], DNA clone characterization [77], earthquake wave propagation simulations [26], plasma edge simulation in fusion physics [50], etc. The need for dynamicity solutions in this scenario increases as the experiments to be performed demand large amounts of computational resources and take a long time to finish. In these situations, it becomes more important to run several alternative experiments at the same time and to introduce changes at certain points without loosing the already valid results. It can also be considered in relation with the development of scientific workflow prototypes and their interactive debugging [29] .

3.2 Scenario B: Ill-defined Problems

Many times, during scientific work, it is not possible neither convenient to try to preview all the possible tasks or parameters in advance. By the contrary, the tasks to be performed or parameters to be used at a certain point may be so dependant on the results provided by the previous tasks that it does not make any sense to try to preview them. At this point, the most desirable solution is to enable the specification of new tasks and their interdependencies or the parameters when the previous results are known.

A clear example of this ill-definition can be found in the area of mathematical solvers. In the climate simulation problem described in [51] the proposed workflow chooses optimized algorithms based on performance measurements to identify a set of resources that fulfill a given high-level function, such as a matrix multiplication. The task of conducting the matrix multiplication is specified at the time the workflow is specified. However, its instantiation and the appropriate selection of which algorithm to choose are conducted during runtime. It is also worthy to mention the Grid-TLSE project [8], which aims at designing an expert site that provides an easy access to a number of direct solvers for sparse linear systems. Sparse direct solvers are very different in practice since they often use different algorithms with their own control parameters. Multiple parameters interfere for efficient execution of a sparse direct solver: ordering, amount of memory, libraries available, etc. This project provides assistance in choosing the right solver and appropriate values for its control parameters in accordance with the characteristics of the problem that

needs to be solved. There is a wide range of applications requiring these methods: seismic imaging [67], cosmological simulations [19], fusion plasma simulations [29], air quality forecasting [84], etc.

In addition to the previoues cases, scientists typically divide up overall tasks into smaller sub-tasks, each of which can be considered to be an individual step in an experiment [27]. They have a clear idea about the global tasks, but the way in which they should be performed by using more specific sub-tasks is not clear. The decisions they make about which steps to take next are based on the latest available information. This is defined as "*component substitution*" in ICENI [57], or "*frame/dynamic embedding* in Kepler [62]. A workflow may need to be dynamically designed in the sense of looking at the results of the initial steps before a decision can be made about how to carry out later analysis steps.

Strictly, the previous problems do not require a dynamical solution, as it could be supported by a solution based on the *Exclusive Choice pattern* [2], that enables to describe alternative workflow paths. Anyway, if the number of alternative paths is very high this is no more a valid solution.

3.3 Scenario C: Event Driven Applications

Nowadays, the application of computational technologies to new problems and the need to get more accurate outcomes demand the use of updated (and in some cases real-time) data inputs. In many scientific applications data acquisition use to be a key part of the process and data sets are being progressively produced [37]. The *Dynamic Data Driven Applications Systems* (DDDAS) concept entails capabilities where application simulations can dynamically accept and respond to field-data and measurements, and/or can control such measurements [24]. This synergistic and symbiotic feedback control-loop between simulations and measurements goes beyond the traditional control systems approaches. In some cases, the simulation can determine how and when the measurements should be made. In other cases, the data or events received can provoke changes in the tasks to perform. For example, in severe storm prediction, data analysis agents may examine radar data searching for specific patterns [36]. Depending upon the specific pattern of events, different branches of a storm prediction workflow may be enacted which may require that significant computational resource be made available on-demand. Should the storm intensify or should resource availability change, the workflow must adapt.

A more simple example, but also showing this event driven behavior, is depicted in figure 4. It is an actual scientific workflow that captures the operation of an experimental study in the Soil Sciences Department of the University of Wisconsin [3]. The objective of the experiment is to produce daily forecasts of near-surface temperatures in cranberry bogs in Wisconsin. These forecasts give cranberry farmers advance warming of over-night frost conditions, so they can take action to protect their fields from frost damage.

1. Around noon each day, satellite and ground-based meteorological observations are processed in the Atmospheric Sciences Department of University of Wisconsin, generating a 24-hour weather forecast at several heights in the atmosphere for the whole United States.

2. This US forecast is fed into a Bog Forecast Extraction program that extracts forecasts for points that are 25 meters above specified cranberry bog locations.

3. These forecasts are sent to the Soil Science Department where they are processed by CranEB to derive a forecast for the level of the cranberry wines.

4. Later in the day, as new weather observations become available, the initial 25m bog forecast can be updated:

   - Scaled CranEB output forecasts are compared with new observed weather conditions in a package of statistical routines.
   - Appropriate corrections to the original 25m bog forecast are determined, and CranEB is rerun.

   With this feedback mechanism, the canopy-level forecast is updated continuously through the day.

5. The files generated by CranEB are fed into the Devise Visualization tool that generate GIF plots of canopy temperature vs. time. These plots are then published on the Web, where they can be readily accessed by cranberry farmers throughout Wisconsin.

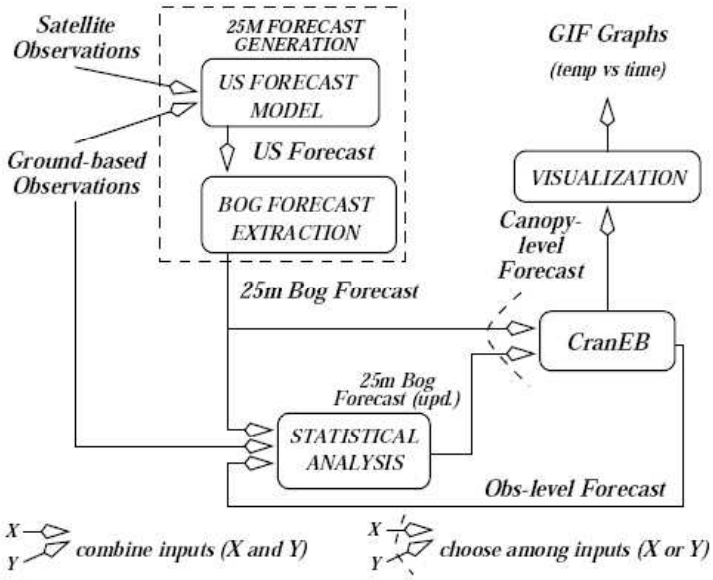

Figure 4: The cranberry workflow(image taken from [3])

Another well-known example of this scenario is found in the LEAD project [28, 33]. Streaming observations from external sensors (e.g., radars) are mined for specific features by a persistent data mining agent. If an event is detected, the system automatically launches a weather forecast. As the forecast output is produced, it is directed to the same data mining engine to identify specific features that, if found, might be used to re-task the radars. The process then repeats in a closed-loop fashion.

In general, this kind of scenario can be found in relation with nature phenomenons that require an accurate and just-in-time response, such as [23]: medical intervention and treatment, environmental and natural resource management (e.g. forests fires, hurricanes, contamination spread), urban traffic management [40], wheather analysis and forecasts, etc.

3.4 Scenario D: Processing Large Amounts of Data

The analysis of large amounts of data is becoming commonplace in many scientific endeavors. In addition, they involve large teams of scientists and technicians, and engage in experimental methods or procedures that take long times to complete. Despite the computational and data management capabilities of new systems are increasing very fast, the demands of scientific problems increase faster. Therefore, efficient data management and computational mechanisms that enable to obtain the maximun performance from existing resources are demanded. For example, parallelization and distribution approaches play a main role towards efficiency as they allow the use of numerous resources. At this point, dynamic behaviors in scientific workflows are also conceived to facilitate such parallelization and distribution.

A clear example of this scenario can be seen in the Climateprediction.net[8] project [5]. It is a distributed computing project inspired by the success of the SETI@home[9] project [9]. Users' computers download a model of the earth's climate and run it for approximately fifty model years with a range of perturbed control parameters. Then, the produced results are returned to one of many upload servers. The data of these models creates a data set that is distributed across many servers in a well-defined fashion. This data set is too big to transport to a single location for analysis, so it must be worked on in a distributed manner. The challenge arises because the number of pieces this dataset is split into varies for a range of reasons, including the addition or removal of servers from the experiment and changes to the location or sub-division of data.

In order to derive results, it is intended that users will submit analysis functions to the servers holding the data sets. Users are unable to ascertain how many servers a given subset of this data that they want to analyze spans, nor

[8] Climateprediction.net Web site at http://www.climateprediction.net/

[9] SETI@home Web site at http://setiathome.berkeley.edu/

should they care. Their interest is in the information they can derive from the data, not how and where it is stored. An example of user function is the average temperature of a given set of returned models [39]. If these temperatures spans a servers, this calculation can be described in a way that could be used for distributed computing as:

$$y_0 = \sum_{i=0}^{n_1-1} x_i$$
$$z_0 = n_1$$

$$y_1 = \sum_{i=n_1}^{n_2-1} x_i$$
$$z_1 = n_2 - n_1$$
$$\dots$$
$$\dots$$
$$y_{a-1} = \sum_{i=n_{a-1}}^{n_a-1} x_i$$
$$z_{a-1} = n_a - n_{a-1}$$

$$\overline{x} = \frac{\sum_{i=0}^{a-1} y_i}{\sum_{i=0}^{a-1} z_i}$$

Each subset of the data set has a computation performed on it, with the results used by a final computation to produce the overall average.

There exist more common examples of dynamic behavior demands related with the processing of large amounts of data in scientific problems [74, 37, 68, 58]. Datasets in scientific workflow may contain static or dynamic (unknown at composition time) number of data elements. In this way, several data collection operators to manage dynamic data collections in scientific workflow initiatives are described. In the ASKALON Grid environment several constructs have been proposed for the distribution of data collections [73]. A data collection is an ordered list of data elements, i.e. a one-dimensional array. The problem to solve is how to map a one-dimensional array of data elements (a data collection) to a number of parallel loop iterations. They propose four distribution constructs: BLOCK, BLOCK(S), BLOCK(S,L) and REPLICA(S). These four constructs are processed at runtime to determine which elements of a data collection are distributed onto which iteration. They involve dynamic behaviors in accordance with the number of data elements and the number of parallel iterations. Similar operators can be found in the JOpera system [69] (e.g., *split*) and Kepler [58].

The authors of JOpera also introduce another example of dynamism related with data changes in scientific workflows [68]. It is about the pipelined execution of several tasks, see figure 5. As opposed to iterative execution, where each data element would have to go through the entire sequence of tasks before the next data element of the vector is processed, with pipelined execution the elements are streamed through the workflow. Similar to the situation described previously, the set of input elements may be know in advanced or new elements may appear while the pipeline is running. The application of a parallelization construct can reduce the overall execution time of the workflow by a factor proportional to the length of the sequence of pipelined tasks. It is particularly interesting the superscalar execution semantics, as it requires to dynamically create additional task instances whenever they are needed to avoid collisions. Thus, if a new input data element is available for a busy task, another instance of the same task is created in order to process the new data element in parallel.

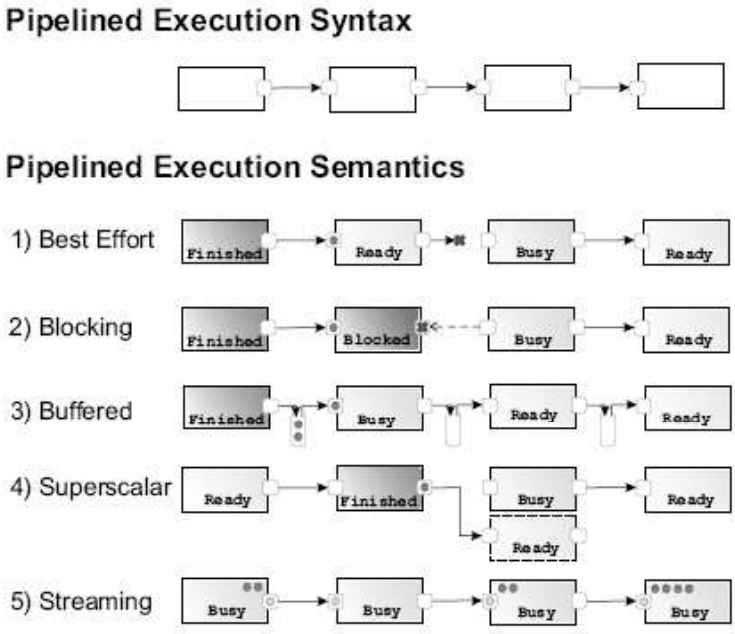

Figure 5: Informal representation of different pipelined execution patterns (image taken from [68])

3.5 Scenario E: Tackling with Real Resources

Scientific workflows usually involve the performance of intensive computations with large amounts of data. During workflow design, appropriate resources to perform computations and manage data are described in a decoupled way, enabling the performance with different resources (e.g., Grids). This is achieved by specifying the functional and non-functional requirements of the resources required to perform each task in a more or less "ideal" way. Nevertheless, final computations and data management are going to be performed not in "ideal" recourses, but in real ones. Real resources involve issues such as availability, capacity and in general performance limitations that may prevent the execution of scientific workflows as they were planned. Moreover, resources can come and go [85, 71], because of failure, local policy changes, overloaded resource conditions, etc. Many of these issues fall into the domain of system and network administrators, who must design infrastructure to provide redundant components. Anyway, solutions involving the dynamic change of scientific workflows are also considered.

A first dynamicity solution to cope with this scenario is the mapping or allocation between workflow specification and resources. The mapping needs to be performed dynamically, adapting the description of the workflow in accordance with the state of the resources at each moment [87]. This issue is also related with the scheduling of workflow tasks, to get the better performance from the resources available. In [72] they use a high intensive computation application to test a dynamic scheduling algorithm against a genetic algorithm.

An example of an scientific workflow environment where execution problems are considered is described in [4]. It is about the e-HTPX project, a distributed computing infrastructure designed for structural biologiest to remotely plan, execute and monitor protein crystallography experiments. The services developed for this project define the e-HTPX Pipeline (or workflow) illustrated in Figure 6, where the following stages are proposed: Protein Production; Crystallization; Data Collection; Phasing (data processing); Solution of digital protein structure model; and Submission of protein model into public database. The workflow covers protein crystallization, delivery of a protein crystal to a synchrotron radiation facility, data collection using X/ray diffraction methods and HPC data processing services. Remote access to these services is implemented by a collection of Web services. Unexpected errors can occur in this infrastructure mainly due to the physicality of the tasks. For example, during data collection if a sample-changing robot fails to operate correctly or a crystal sample is damaged in transit. Because these errors are physical in nature, they are difficult to forecast. Furthermore, because these types of error cannot be anticipated by the workflow-engine they are difficult to handle. In such cases, it may be necessary to restart the workflow form a specific breakpoint and

roll back to a point where all prior activities were successfully completed.

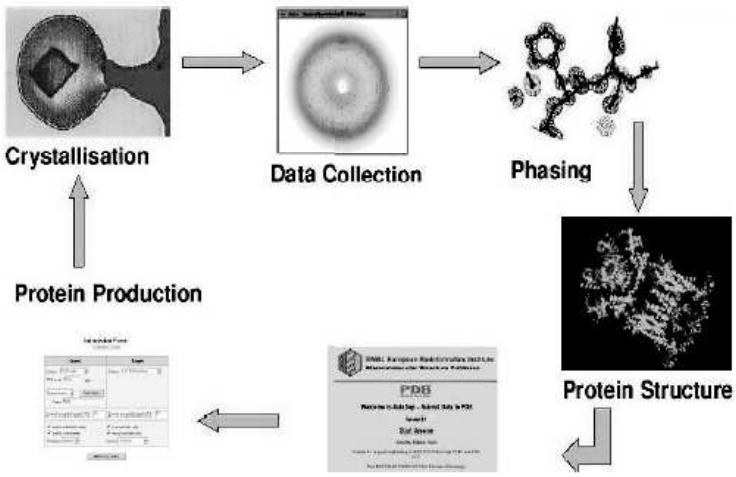

Figure 6: e-HTPX Protein Crystallography workflow (image taken from [4])

Finally, in certain domains workflows are not developed in isolation, but they interact with other resources. In [47] the following situation is described: "suppose that a certain business unit of a company is awaiting the results of a Grid workflow that is taking more time to complete than expected. In such a case, it could be useful if the workflow could be modified while running in order to get it to finish faster (e.g., by removing certain parts of the workflow that are not considered essential to obtain the results needed)".

4 Dynamicity Requirements in Scientific Workflows

The previous section introduces some general scientific scenarios involving the use of workflows and large amounts of computational and information resources. Supporting the scientific work in those scenarios is an important challenge as it requires the provision of dynamicity solutions. The literature has introduced many requirements [36, 66, 40, 47, 55, 50, 87, 77]: appropriate user interfaces that provide scientist with tracking information about the scientific process development, facilities to change the parameter space, capabilities to change the workflow activities, etc. This section introduces a classification for these requirements. A first grouping distinguish between two different kinds of requirements: *change requirements* (CR) and *usability requirements* (UR). The first ones are about issues that should be able to change during scientific workflow execution. The second ones are about issues required to facilitate the performance of such changes. Next sections describe the requirements considered at each group.

4.1 Change Requirements

This section introduces a generic analysis about the issues that could be required to change in scientific workflows dynamically. It is based on a previous proposal in Grid workflows [51]. In accordance with this proposal, the elements involved in the instantiation of a scientific workflow model are: a set of Grid resources and/or services' instances, a quality expectation defined by the user(s) and a workflow abstract model. We consider several modifications to this formalization through the generalization of the quality user expectations as *Data Elements*, enabling also changes in input data, and the introduction of *Abstract-to-Concrete Transformations*. In this way, the formalization of a scientific workflow model is as follows:

$$\mathcal{W}_i = (\mathcal{G}_r, \mathcal{G}_s, \mathcal{D}_v, \mathcal{C}_t, \mathcal{W}_m) \quad (1)$$

where

$\mathcal{W}_i =$ Workflow instantiation (also know as *concrete* workflow),

$\mathcal{G}_r$ = Grid resources,
$\mathcal{G}_s$ = Grid services,
$\mathcal{D}_v$ = Data values,
$\mathcal{C}_t$ = Abstract-to-Concrete transformations,
$\mathcal{W}_m$ = Workflow Model (also know as *abstract* workflow).

The concept of dynamically changing workflows and their associated states can be denoted through the addition of a time stamp and an adaptation function a at time T. Hence,

$$\mathcal{W}_i^T = (\mathcal{G}_r^T, \mathcal{G}_s^T, \mathcal{D}_v^T, \mathcal{C}_t^T, \mathcal{W}_m^T) \tag{2}$$

$$\downarrow a^T \tag{3}$$

$$\mathcal{W}_i^{T+1} = (\mathcal{G}_r^{T+1}, \mathcal{G}_s^{T+1}, \mathcal{D}_v^{T+1}, \mathcal{C}_t^{T+1}, \mathcal{W}_m^{T+1}) \tag{4}$$

In order to specify a transformation function, a series of previous workflow instantiations may be considered to determine a future instantiation. Hence:

$$\mathcal{W}_i^{T+1} \leftarrow a(\mathcal{W}_i^{T_0}, \cdots, \mathcal{W}_i^T) \tag{5}$$

where T_0 is the initial time.

Figure 7 shows the identified requirements in accordance with an adapted version of the WfMC reference model [43], as it has also been proposed to analyze scientific workflows [85]. The circled numbers indicate the typical location of the elements involved at each requirement. Next sections include a more detailed analysis of these five requirements.

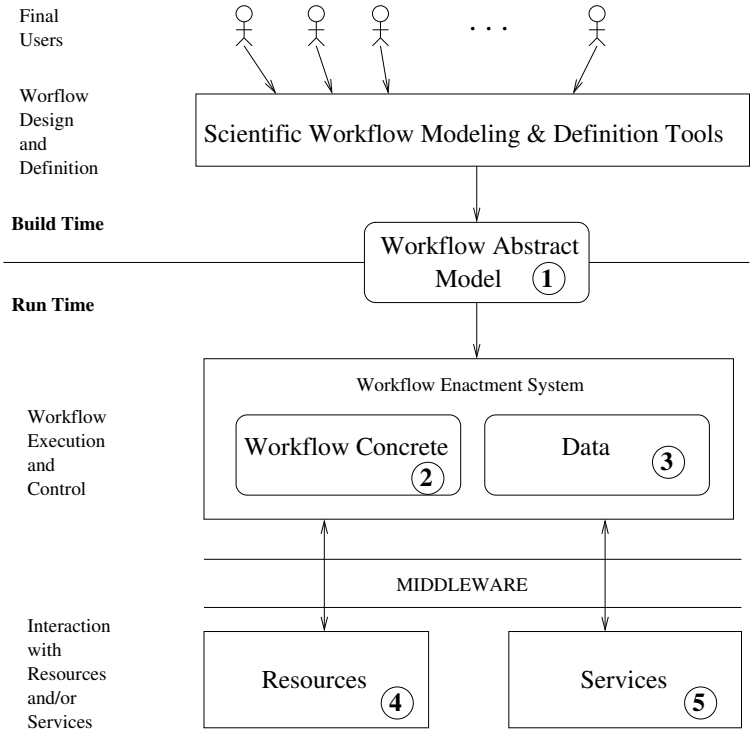

Figure 7: Dynamism requirements Reference Model

4.1.1 CR-1: Changes in the Workflow Abstract Description

The modification of workflow abstract descriptions while they are being executed is a typical dynamicity requirement in scientific workflows. It is associated with "human in the loop" situations as described in scenarios A and B [34, 10, 20, 47, 40, 32, 87]. In some cases, scientists do not want, or do not know how to fully design and construct a workflow specification before executing an experiment. In other cases, different alternatives to proceed with the workflow can be explored. The user may also want to modify the workflow during runtime depending on intermediate results. In real practice, changes can consist in tasks to be added, replaced or removed. It is also possible to to modify the control or data flow/dependencies between tasks. For example, several tasks that could be performed in sequence now can be performed in parallel.

4.1.2 CR-2: Changes in Abstract-to-Concrete Transformation

It is possible to require changes in the Abstract-to-Concrete workflow transformation. As it has been introduced in the workflow life-cycle description, see section 2.3, the workflow descriptions produced by final users use to involve certain abstract tasks that need to be determined before execution. This stage enables to use different methods to achieve a certain goal, providing in this way an adaptation layer that enables to obtain performance enhancements. The scenario D also may require transformation from an abstract to a concrete workflow description. For instance, the ClimatePrediction.net example considers transformations to support the processing of the large amount of data distributed in several servers. Therefore, it is also possible to consider changes in these transformations. This requirement is not usually recognized, as not all the systems involve this stage.

4.1.3 CR-3: Changes in Data

The ability to change the involved data in scientific workflows is a very common dynamicity requirement. Many times in scientific workflows, users need to change certain parameters or input data values dynamically (e.g., user quality expectations [51]), as it is described in the scenario A. Other times, workflow data is not available when the workflow is started, but it is dynamically obtained, as in the scenario C. Moreover, some workflows may act on the data itself, organizing it in appropriate data sets in order to be processed appropriately, as in scenario D. As the number of data inputs is not known during the design time, it is not possible to describe a generic workflow involving a certain number of data elements. Then, the workflow has to adapt to perform the described function at runtime once the final number of inputs is known.

To support this requirement is important that the scientific workflow execution does not need to be re-started since the beginning, but it can use the already valid performed computations. It is also important to be able to relate the results obtained with the data inputs used, in order to facilitate scientists examining the obtained results. For example, it should be facilitated the comparison of the results obtained from the execution of the same workflow with different input data.

4.1.4 CR-4: Changes in the Resources

An important feature in computational systems supporting scientific workflows, usually Grid systems, is the continuous change of resources [40]. The resources available are not guaranteed and may be switched off or altered with time. Moreover, the resources can fail to execute the assigned tasks, and they may be blocked or stalled. Therefore, if someone is using addresses (or names) of such resources to build a workflow, it is important to be aware that such a workflow will only be processable by those particular resources. In order to overcome this obstacle the solution must support an abstract layer in resources' identification. Then, the names used by the workflow designer will not become outdated due to changes in the processing system.

In the case of abstract identification of resources, the workflow has to be composed with the support of automated discovery and assembling of available resources. At run time, the engine should adapt to meet the dynamic changes occurred in the deployed resources. This is more important taking into account that scientific workflows use to involve complex problems, potentially utilizing thousands of resources that can run for several hours, days or even weeks. At this point, it is also necessary to take into account issues that cope with possible problems during the workflow execution, such as self-healing systems [64], load balancing [83], fault-tolerance [70], etc.

In a certain way, this requirement is applied to the whole scientific workflow model and not simply to dynamicity. Anyway, scenarios D and E, introduce specific unexpected situations that require a dynamicity solution. In scenario D,

the management of large amounts and the need to distribute and parallelize their processing requires the involvement of different resources dynamically. In scenario E, it is required the adoption of mechanisms to cope with resource failures and availability constraints.

4.1.5 CR-5: Changes in the Services' Instances

Similarly to the changes in the resources considered in the previous section, it is also possible to need changes in the services' instances. The service-oriented approach to workflow management introduces an abstract layer enabling the use of different instances of a certain service during execution. In this way, it should be possible to change the instance used.

As in the previous case, this requirements is also intrinsic to the scientific workflow model. Anyway, it is considered as a specific dynamicity requirement in relation with scenarios D and E.

4.2 Usability Requirements

This section considers requirements about how changes during scientific workflow execution can be appropriately supported by workflow management systems. This is an important point, because it is necessary to provide suitable functionality to perform changes in a way as simple as possible. Modifications should not only be feasible, but they should be performed with at a "low" cost.

Two different kind of models for introducing changes during workflow execution are identified. On the one hand, a model where the "human user" (e.g., a scientist) is in control. This user has to be provided with an appropriate interface to monitor and control the scientific workflow execution. On the other hand, an autonomous (or self-organizing) model where modifications are performed in a controlled and automatic way following rules or algorithms. The first model is more related with scenarios A and B, while the second model is more related with the scenarios C, D and E. Anyway, the final needs of a certain system use to involve issues from both models. Next sections introduce eight usability requirements. The first four are about common issues in order to facilitate and enable the performance of changes. The next two requirements come from the "human user" model and the last two requirements from the autonomous one.

4.2.1 UR-1: Monitoring

An important part of any long-running scientific workflow is *monitoring* of its progress and outputs [77]. This real-time feedback is necessary both for human users and for autonomous systems to detect any kind of problem during execution. In addition, the monitoring information is necessary for scientists to be able to interpret and validate the results.

The monitoring challenge is to supply the information at the right level of detail in a form that is easily understable by the final user (at this point the differences in user expectations make this challenging). Making this information available requires the collection of data and their processing towards an appropriate presentation. Operations such as aggregation, correlation and filters can be required. The idea is not to provide raw data, but processed data closer to the user expectations. Human user monitoring is usually performed through appropriate graphic visualizations in order to facilitate the understanding of the results [14]. These graphic visualizations should be supported by user interfaces that enable scientists to browse/traverse, query and select information of interest.

Finally, monitoring of the workflow execution should be provided both in real-time and also off-line, as a kind of memory. Recordings of monitoring information are very important in order to validate the results obtained and to compare the results from variants of the same experiment (e.g., using different tasks, input data or parameters).

4.2.2 UR-2: Automatic Control

Due to the heterogeneous and distributed nature of systems supporting scientific workflows execution (e.g., Grid systems) faults inevitably happen, as it was described in scenario E. *Automatic control* is considered to complement monitoring by supporting the detection of certain situations of interest. The final idea is to be able to notify of any fatal failures or long delays. As in the previous requirement, such a notification can be provided to a human user or to an autonomous system that has to take a solving action.

In addition to the typical detection of failures and execution problems, some scientific workflows also use to require automatic controls in order to cope with unexpected external events. Typically, workflows requiring the processing

of real-time data, as in scenario C, need to be aware of new data elements available or any other event of interest. Similarly, many workflows involve the processing of optimization algorithms, consisting on an optimization loop that converges after an number of interactions are processed. The number of interactions is not know beforehand, but it is determined in accordance with an optimization criterion. These situations require the automatic control of certain data elements and/or events.

4.2.3 UR-3: Reproducibility

A general requirement in scientific work is the ability to reproduce results obtained. Just as the results of a conventional lab experiment should be reproducible, computational experiments and runs of scientific workflows should be reproducible, indicating which specific data and tools have been used. Reproducibility is also used when errors are produced and the workflows need to be examined to try to discover the causes.

In the case of dynamic workflows *Reproducibility* becomes a more important requirement, as it is supposed that process and data can be changed during the execution. In this way, information about the changes and modifications applied should be available in order to get a precise knowledge of the tasks performed and the data involved [36]. The challenge is to develop mechanisms to create, manage and capture dynamic workflows so that reproducibility of significant results is possible. In addition, it also should be possible to allow the final user to introduce comments in the workflow results in order to facilitate their later revision.

4.2.4 UR-4: "Smart" re-runs

The "smart" re-runs requirement is about the ability to perform a dynamic change and re-initiate the workflow execution without repeating the already performed work not affected by the change. Usually, it requires to re-initiate the workflow at an already performed stage, allowing to backtrack to a saved state without starting over from scratch. This requirement is also needed in the case of a system failure. In addition, it becomes more important when scientific applications use to involve complex problems, potentially utilizing thousands of resources for workflows that can run for several hours, days or even weeks.

4.2.5 UR-5: Steering

Steering is about the interactive execution of workflows enabling to run, stop, pause and inspect the status of tasks and data [48]. This requirement needs to be combined with the monitoring facility in order to facilitate human users to control the execution. In addition, *Steering* is considered in order to enable the detection and analysis of problems during workflow execution. In this way, it is considered as the typical debugging functionality available in many computer programming environments. At run-time, a user should be allowed to control the progress of the flow execution (e.g., through VCR like operations: play, pause, stop, proceed and operations), to steer the execution of a flow, to introduce breakpoints, to interact with constraints defined in a flow, and in general to change or modify the flow description [47, 34].

4.2.6 UR-6: User Modifications

Most of the time scientists perform experiments by varying the tasks performed or the parameters used, as in scenarios A and B. In addition, some problems that may appear during execution could be solved by performing some modifications. Therefore, execution systems should provide interactive GUIs that enable *human users* modifications to the workflow model and to the data involved are required [77]. In order to facilitate the performance of changes by non-expert users (e.g., scientists), such interfaces should provide a broad set of facilities and mechanisms [27] that enhance the performance of data changes (with appropriate displays for different types of data), assistance for composing scientific workflow descriptions, etc.

It is also important to notice that many scientific workflows require user decisions (e.g., to select a region of interest in an image) and interactions at various steps [55, 66]. For example, an improved version of PIW (example described in scenario A) allows the user to inspect intermediate results and select and re-rank them before feeding them to subsequent steps.

4.2.7 UR-7: Adaptations from the Workflow Description

Many times changes to be performed in a workflow can be planned during design time or they can be previewed (despite a clear specification is not performed). Therefore, the workflow language should provide constructs to support the description of scientific workflows including possible adaptations. A main need is the feasibility to include abstract tasks enabling their detailed specification during run-time, as in scenario B. In addition, it should be possible to use conditions and event-controls (as in scenarios C and E) to decide the use of certain parts of a workflow description during run-time. This requirement is also found related with the performance of parametric studies. In this case it is required that the same workflow is tested against a range of parameters [38].

This requirement is also present in the scenario D. In this case, the scientific workflow description has to be adapted in accordance with the number and distribution of data. To do it, it can be required to break down the workflow in several parts that can be executed in different servers. A common solution is to the divide large data collections into several small chunks in order to allow a similar parallelization. In this case, the number of chunks and their size have to be determined dynamically during run-time.

This requirement could be considered not as a real dynamicity requirement. As it has been described, it can be previewed before the workflow execution and, in this way, no modifications have to be performed during run-time to the workflow abstract description. Nevertheless, in more cases, it is required to change the workflow instance (abstract-to-concrete transformations, data, resources or services' instances) and, in this way, we consider it as a dynamicity requirement.

4.2.8 UR-8: Adaptations from the Workflow Execution Environment

The scenario E indicates that the resources available to execute the tasks involved in scientific workflows are changing continuously. In other way, a main basic need is to execute the same workflow in different computational systems. In addition, scientific workflows are usually executed under efficiency constrains, requiring the use of computational capabilities in a flexible way in order to obtain performances as high as possible. Therefore, the workflow execution system needs to perform its work in a dynamic way, involving issues such as: the assignment of resources to tasks, the transfer of data between tasks, etc. The execution system should perform these adaptations in an autonomous way, without requiring the user interactivity neither the description of special construct in the scientific workflow specification.

4.3 Analysis of the Identified Requirements

The dynamicity requirements introduced in this section are intended to capture the common needs in this field in a generic way. Challenges in approaching the support of dynamicity requirements stem from the fact that user expectations and needs vary greatly. Many users typically want to be able to focus solely on the scientific aspects of the problem. However, other users may want to look deeper into the workflow execution and guide the mapping process, perhaps indicating preferred compute resources and data sources. Therefore, the establishment of specific requirements depends of the concrete problem to a long extent.

It is important to notice that these requirements can not be considered independently, but they are co-dependent. This is clearly evident in the case of monitoring and user interaction. In order to support the second requirement it is quite obvious that the first one should be solved already. Anyway, by distinguishing between the two requirements it is possible to approach each one step by step, focusing the attention in one concern at each time and simplifying the whole problem. In addition, requirements such as monitoring can be considering in a scenario without requiring user modifications.

Table 1 shows a summary of the requirements and their relationships with the proposed scenarios. It is quite obvious that the different requirements are distributed in the several scenarios. Anyway, in general it is possible to distinguish between two great sets of requirements. In the one hand, the requirements oriented towards the human user operation. In the other hand, the requirements oriented towards autonomous execution systems. Both sets share the first four usability requirements. Anyway, it is quite common to conceive solutions involving requirements for each group.

Table 1: Relationship between dynamicity requirements and scenarios in scientific workflows

Scenario	A	B	C	D	E
CR-1: Workflow Abstract Description	X	X			
CR-2: Abstract-to-Concrete Transform.		X		X	
CR-3: Data	X		X	X	
CR-4: Resources				X	X
CR-5: Services' Instances				X	X
UR-1: Monitoring	X	X			
UR-2: Automatic Control	X	X	X	X	X
UR-3: Reproducibility	X				
UR-4: "Smart" Re-runs	X				X
UR-5: Steering	X	X			
UR-6: User Modifications	X	X			
UR-7: Adaptations from the Wf. Language			X	X	X
UR-8: Adaptations from the Wf. Execution Environment				X	X

5 Existing Techniques to Support Dynamicity

This section reviews some existing techniques used to support dynamicity in current scientific workflows. These techniques are classified in the following two groups:

- Proposals at the Modeling Language Level. These techniques are modeling languages mechanisms used to create workflow specifications. They have to be applied during the design-time, but they also require appropriate processing during run-time.
- Proposals at the Execution Level. These techniques are carried by the executing system. Contrary to the previous case, these techniques do not require any action during the design-time and they have to be performed transparently for the user during run-time.

Next sections introduce some of the main techniques in accordance with this two level classification.

5.1 Techniques at the Modeling Language Level

The dynamicity techniques at the modeling language level involve issues about the general structure of the language and about specific constructs. This section analyzes the following four techniques: (i) modular and hierarchical design of scientific workflow descriptions; (ii) abstract specification of task requirements; (iii) semantic task description; and (iv) task placeholders.

5.1.1 Modular and Hierarchical Design

Modular design is related with the aggregation of workflow tasks into modules that are operated as "back boxes", each having its own variables, constraints, event handlers, etc. In real practice, modularization is usually derived into a hierarchical structure, enabling that each module at a certain level is broken down into several modules at a more depth level [40]. These modular and hierarchical designs are appropriate to control the complexity of the workflows and to support some of the dynamicity requirements. In the one hand, by modularizing workflows, workflow components can be made sufficiently scalable and reliable to serve as reusable units. In the other hand, failure in one module can be controlled and managed locally, facilitating the performance of required changes without affecting to the whole workflow.

The hierarchical design approach also facilitates the refinement of abstract workflow descriptions at runtime. This idea is complemented with the consideration of the workflow enactment in an interpreted way, in contrast with the complete compilation of the workflow abstract description before the execution. This allows the just-in-time translation of the modifications from the abstract description to the concrete code, facilitating the performance of changes.

5.1.2 Abstract Specification of Task Requirements

This technique involves the description of workflows without specifying a binding of each task to a concrete resource or service instance, so the bindings can be added at run-time [40]. *Performance Contracts* are a refinement of this technique, involving the definition of the computational needs for each task. The reason for this technique is found in the dynamism of the executing (e.g., Grid) environment: the resources available are not guaranteed and may be switched off or altered with time. Therefore, if someone is using addresses (names) of such resources to build a workflow she risks the possibility that in the future the workflow will no longer be executable. In order to overcome at the level of this obstacle, the solution is to provide an abstract layer in resources' identification. The key idea is that the names used by the workflow designer will not become outdated due to changes in the environment. In case of the service approach, in order to abstract away from the certain instances of services published in the Grid, oa concept of an abstract service which could be dynamically mapped on the realization(s) available at the execution time is proposed.

5.1.3 Semantic Task Description

This technique involves the semantic description of a task, but not the concrete details about what has to be performed. In this way, during run-time the task can be developed in accordance with the execution of different processes, depending on the previous results, the availability of resources, the time-constraints, etc. Semantic mechanisms are proposed that derive how tasks have to be developed eventually. This technique is also used to facilitate the dynamic composition of workflows, helping the user to make decisions and guiding the user in developing semantically correct workflows.

5.1.4 Task Placeholders

Task Placeholders can be defined as "empty tasks" that can be included in a workflow specification during design-time. In addition, it is explicitly stated that the task specification has to be performed during run-time before its execution. In this way, complex problems are encapsulated and their solution can be delayed. This concept of placeholder task has been proposed in Business workflows [22] and in some scientific workflow systems, such as Kepler where Placeholders are named as Frames [62].

5.2 Techniques at the Execution Level

The proposals at the execution level involve automatic actions performed by the execution system without the need of human control. Many of the existing techniques are related with fault tolerance and they are analyzed in the following section. There exist other techniques, such as refinement and interpreted execution, that can be related with mechanisms described in the modeling language section. Finally, specific runtime techniques as checkpointing and provenance are also analyzed.

5.2.1 Techniques Related with Fault Tolerance

The existing literature distinguish between task- and workflow-level fault tolerance [85, 44]. The former is about faults that happen during the execution of single tasks while the latter manipulates the structure of the workflow to deal with faults dynamically.

At the task level, upon detecting a failure, the task is rescheduled to either the same or to another resource for another try. Task-level techniques are:

- *Retry*. It simply tries to execute the same task on the same resource.
- *Alternate Resource*. The failed task is submitted to other resource.
- *Checkpoint/restart*. Moves the failed tasks transparently to other resources, so that the task can continue its execution from the point of failure.
- *Replication*. The same task is run simultaneously on different resources to ensure task execution provided that at least one of the replicas does not fail.

At workflow level redundancy, data and workflow replication are proposed as strategies to tackle with faults. Workflow-level techniques include

- *Alternate Task*. Another implementation is executed of a certain task if the previous one failed.
- *Redundancy*. Multiple alternative implementations of the task are executed simultaneously.
- *User-defined Exception Handling*. It allows the user to specify a special treatment for a certain failure of a task in the workflow specification.
- *Rescue Workflow*. Failed tasks are ignored and then a rescue workflow is executed to generate a report about failed tasks.

In order to detect faults *Performance Contracts* can be monitored to control whether tasks are executed properly or whether they should be migrated.

5.2.2 Workflow Refinement

The tasks involved in a workflow may be performed using different algorithms, each of which may have multiple implementations (e.g., different execution architectures: Linux vs. MS Windows). The mapping of tasks to implementations takes place at the run-time and can be considered as part of the *Abstract-to-Concrete transformation* stage. At this mapping it is possible to consider several manipulations in order to produce a more optimal workflow [57]:

- *Re-ordering of tasks.* Changing the order in which tasks are intended to be performed.
- *Insertion of additional tasks*. For example, compression of image files.
- *Workflow Substitution*. For example, in accordance with conditions of semantic equivalence.
- *Task Substitution*. As tasks can have multiple implementations, which may be suited better to different input types.

5.2.3 Interpreted Execution

Two main approaches can be observed towards the execution of scientific workflows: *compiled* and *executed*. In the *compiled approach*, the workflow description performed in a certain modeling language is completely "translated" to a runable form before initiating the execution. In the *interpreted approach*, the workflow specification is not "translated" into any runnable form, but it is read and processed step by step. As in other areas, the compiled approach is faster, while the interpreted approach gives more opportunities to perform dynamic changes. Clearly, in the interpreted approach the changes to parts of the abstract workflow that have not been executed at a certain point can be performed for free. Nevertheless, the adoption of an interpreted approach is not enough to support the dynamicity requirement, because it is also necessary to take into account the execution state. If the part of the workflow specification to be changed has been executed already, more actions need to be performed in order to re-start the workflow at a previous stage by modifying the execution state. A key point is that the workflow execution should not be re-started from the beginning, but in a point as close as possible to the part changed, taking advantage of the already performed tasks. This is more important in long-lasting workflows.

Mixed approaches between compilation and interpretation can be found, as for example "lazy compilation" in which the compilation is performed as late as possible. In this case, only small portions of the workflow are "translated" ahead of time and then executed before progressing further. An application of the lazy compilation is to optimize the workflow execution with respect to a given criteria [40]. This dynamism allows for as-late-as-possible decisions to allow the scheduling based on the most recent resources' information.

5.2.4 Checkpointing

Checkpointing allows to backtrack (in the case of a change or event a system failure) to a previous saved state without starting over from scratch [55]. Usually, it is also related with the specification of "hotspots". A hotspot is a marked stage in the workflow specification that indicates when the state of the workflow instance has to be persisted.

It is possible to distinguish between light- and heavy-weight checkpointing. Light-weight checkpointing saves only the current location of the intermediate results, not the data itself. It is fast, but restarting can only work as long as data its available at its original location. Heavy-weight checkpointing saves all the intermediate data to a place, where it can be kept as long as it is needed.

5.2.5 Provenance

A main requirement in scientific workflows is reproducibility, as it has been shown in a previous section. Many workflow modeling languages enable to indicate in the workflow description what intermediate results have to be stored to support such a requirement. Nevertheless, this is not an appropriate solution as it demands an important effort from the final user. Automatic solutions to capture and manage the results generated, the algorithms applied, the input data and parameters used are proposed as *Provenance* techniques. Many authors perceive provenance [36, 59] as a crucial component of workflow systems that can help scientists to interpret, validate and ensure reproducibility of their scientific analysis and processes.

6 Dynamicity in Current Scientific Workflow Systems

This section analyzes the dynamicity support available in some of the most well-known scientific workflow systems. This review has been performed in accordance with the information available in the included references.

6.1 Askalon

Askalon[10] builds upon its own XML-based workflow language called *Askalon Grid Workflow language* (AGWL) [30, 31, 73]. This system includes an UML based workflow composition tool, named as Teuta, to enable the authoring of scientific workflows and the automatic generation of AGWL XML documents.

AGWL is a resource-based workflow specification language. A scientific workflow is seen as a collection of computational tasks to be processed in a well-defined order to accomplish a specific goal. AGWL enables the specification of scientific workflows at an abstract level [40]. It works at a high-level of abstraction without dealing with implementation details. At this high level, tasks correspond mostly to computational entities and the specification of its input and output data. There is no notion of the implementation of tasks, how input data is delivered to tasks, how tasks are invoked or terminated, etc. *Performance contracts* for each task can be specified to indicate the computational resource needs. In addition, it supports hierarchical workflow composition through import elements, that enable to import a (sub-) workflow into another workflow. In other way, AGWL follows a data-driven modeling approach [30], but it also includes control-flow links. It adds complex constructs such as parallel loops with pre- and post-conditions, synchronization mechanism, conditional branches (e.g., switch, if/then/else), event-based selection of activities, etc. It also includes several data collection operators, enabling to break down a large data set into several small parts. In some operators the size of the parts is known during the design-time, while other operators enable to produce parts whose size is determined "dynamically" during the run-time [73].

The Askalon infrastructure transforms the AGWL XML documents to a *Concrete Grid Workflow Language* (CGWL). CGWL adds information needed for the execution of the workflow. Particularly, it is indicated how tasks are actually implemented, how they are deployed, invoked and terminated; how data is transferred; etc. The output of the CGWL "compilation" is also an XML document that is used to execute the workflow.

Askalon works on a Grid Resource Management System callled GridARM (*Askalon's Grid Resource Management System*). This GridARM System comprises aspects from resource matching to reservation mechanisms, in order to provide a scalable resource management. An Askalon *Scheduling Engine* queries the GridARM to receive information about how activities can be deployed on individual resources. The *Scheduling Engine* is responsible for actual scheduling of the workflows. It receives the CGWL XML document and it queries the resource manager (GridARM) to map the tasks to resources. Askalon has developed a light-weight just-in-time scheduling strategy [75]. This strategy has been used to support dynamic scheduling of scientific workflows in Grids [72] adapting to the existing resource availability and conditions. It uses an hybrid approach to schedule single workflow applications based on the following two algorithms: (i) static scheduling, as a solution to an NP-complete optimization problem; and (ii) dynamic scheduling,

[10] Askalon Web site at http://www.dps.uibk.ac.at/projects/brokerage/

based on the repeated invocation of the static scheduling algorithm at well-defined events depending on the Grid resource load variation. The scheduler implements different fault-tolerance and fair-sharing policies. For instance, the jobs failed can be restarted or submitted to other resources. It checks *Performance Contracts* during run-time in order to detect any possible problem. These techniques are complemented with the option to use *checkpoint* and *roll back* solutions. In addition runtime steering and dynamicity are mentioned in [71], but this functionality is not detailed.

6.2 Java CoG Kit - Karajan

The Java *Commodity Grid* (CoG) Kit[11] is a layered software architecture and framework to facilitate the use, programming and management of Grids.As part of this framework, a workflow component repository is proposed [82]. A workflow component is a piece of a process description, namely: a task. This repository is intended to support dynamically changing workflows as the workflow engine follows an interpreted approach. The framework allows the dynamic inclusion of workflow components through the use of an *include* statement, that enables to fetch the component from the repository and to evaluate the contents at runtime. Due to the ability of the workflow system to dynamically load workflow components it is possible to design dynamically changing workflows dependent on the state. This is an event-based solution.

Karajan [51] is the Grid task management language and execution engine developed as part of this Java CoG Toolkit. It aims to provide the scientific community with an easy-to-use tool to define and manage complex jobs on computational Grids, while offering some advanced features, such as dynamic execution. The workflow language introduce the concept of element.

6.3 Kepler

Kepler [55] is a open-source graphical workflow system. Kepler is build upon the Ptolemy II framework, developed at the University of California at Berkeley, that is a mature dataflow-oriented workflow architecture. The Kepler project has extended Ptolemy II towards scientific workflows through adding support for Web Service invocations and access to Grid resources. The approach is therefore both resource based and service based.

Ptolomy II [52] is a Java-based environment for heterogeneous modeling, simulation and design of concurrent systems. The goal of Ptolomy II is to enable the building of models based on the composition of components called *actors*, namely: tasks. Actors are encapsulations of parametrized actions performed on input tokens to produce output tokens. Inputs and outputs are communicated through ports within the actors. They provide the common abstraction used to wrap different types of software components, including sub-workflows (hierarchical workflows), Web and Grid services. The interaction between the actors is defined by a *Model of Computation* (e.g., continuous time, discrete event, synchronous dataflow).

Processes in Kepler are represented as *Directed Acyclic Graphs* (DAGs) in accordance with *Modeling Markup Language* (MoML). This language includes both data and control constructs (e.g., loops). Kepler allows the actor communication (dataflow) concerns to be separated from the overall workflow coordination, which is defined in a separate component called a *Director*. Concrete actors can be hierarchically composed and orchestrated by different directors (schedulers).

The analysis of dynamicity support in Kepler is quite difficult. The existing literature has been performed by different authors that maintain different points and introduce unrelated proposals. Next, several proposals related with dynamicity are introduced:

- In Ptolemy II there is the concept of mutation that enables changes in workflow structure at runtime.
- Accordingly to [40] Kepler does not bind workflow descriptions directly with the realizations of their building elements, but in practice it works as this. A common assumption in Kepler is that workflow is static and must be completely specified before orchestration. In this way, the performance of changes to the abstract workflow during run-time is not allowed.
- In [63] an extension to Kepler is proposed in order to support dynamic binding of resources, separating the design time from the run-time. Nevertheless, they do not consider the change of the workflow definition during the run-time and it is a very initial proposal.

[11] Java CoG Grid Wiki Web page at http://wiki.cogkit.org/index.php/Main_Page

- In other way, Kepler supports "human in the loop" by mailing users using the "*mail*" actor (a specific Kepler task) to notify them about specific situations and it also has "*form*" actors (another kind of specific task) to request users for inputs during flow execution. In addition, the execution system has start, pause and stop buttons ("VCR"-like controller) for the user to control workflow execution during execution of the workflow.
- A task placeholder solution is proposed in [62, 15] to model abstract tasks and to enable the dynamic embedding during run-time in an automatic way. Tasks can be partially specified via abstract actors called *Frames*. The behavior of a *frame* is determined at runtime by embedding dynamically an appropriate component.
- Higher-order controls that allow a functional approach to workflow design are introduced in [58, 62]. These controls are similar to the Askalon data collection operators involving collection-aware and list comprehension operators. The goal is to hide the controls that are peripheral to the goal of the workflow. These controls are oriented to the sole purpose of routine manipulation of complex scientific data (e.g., collection or list-based input) or for various exception handlings. The collection-aware tasks allow to simplify the workflow (as control flow controls can be eliminated) and allow reuse of the same workflow with varying nested complex data.
- A provenance system for Kepler with the support of "smart" re-runs is proposed in [7].

6.4 KW-f Grid

The *Knowledge-based Workflow System for Grid Applications* (KW-f Grid[12]) proposes a multi-level abstraction and semantic-based solution to facilitate both the automatic specification of scientific workflows and the de-coupling between tasks and resources or services.

The project includes the development of several languages: GWorkflowDL [6, 40, 61] and GJobDL [42], GResourcesDL and GTaskDL. The GWorkflowDL is a solution based on *High-Level Petri Nets* (HLPN). Transitions are considered as service operations and tokens as workflow data. This model is intended to support control flow and data flow, the dynamic binding of services and the change of the workflow definition during run-time. Therefore, it is proposed to solve the problems of changing resources and changes in the workflow description related with abstraction.

This proposal involves multiple levels of abstractness (cf. figure 8). The consideration of multiple abstraction levels, inherited from the HLPN basics, helps to express the application logic on sufficiently high level for the end user to comprehend it and for the workflow to be reusable. A first level of abstractness is related to the automatic specification of scientific workflows in accordance with an ontology. The user indicates the result in which she is interested. The use of the ontology enables to obtain the workflow specification to produce such a result. This is a solution that can be applied dynamically to help the final users to specify certain sub-workflows during runtime, as these users usually are not experts in workflow composition.

The *Grid Workflow Execution Service* (GWES) [41] is intended to support dynamic and interactive execution and visualization of distributed workflows. The workflow mapping is dynamic and just-in-time. The dynamism is supported as it is possible the runtime refinement of the workflow elements that are invoked and the lazy scheduling strategies that use the more recent information available.

6.5 Pegasus

Planning for Execution in Grids (Pegasus[13]) [25] is a workflow mapping engine developed and used as part of several NSF ITR projects (GriPhyN[14], NVO, and SCEC-CME). It is focused on data intensive applications related with atomic physics, astronomy, biology, etc.

The scientific workflows can be modeled using a *Virtual Data Language* (VDL[15]) at a high level of abstraction. It enables to specify dependencies among tasks using basic elements, such as: transformation, derivation and innovation. The workflow specification is performed using a tool called Chimera. It provides a catalog of programs that can be used to compose workflow specifications and track the execution automatically. In addition, workflow specifications can be composed automatically by indicating the desired output result [27].

[12] KW-f Grid Web site at http://www.kwfgrid.eu/
[13] Pegasus Web site at http://pegasus.isi.edu/
[14] GriPhyN Web site at http://www.griphyn.org/
[15] VDL Web page at http://www.ci.uchicago.edu/wiki/bin/view/VDS/VDSWeb/WebMain

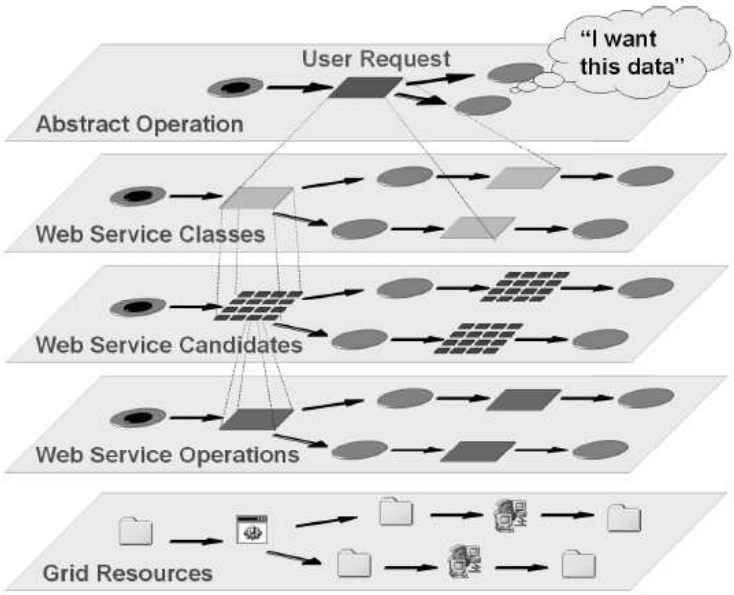

Figure 8: The abstraction levels that are supported by the GWorkflowDL and the GWES (image taken from [41])

The Pegasus system receives a VDL document with a workflow specification and generates a concrete workflow. The resource brokering is performed at this stage, given the description of the resources required by a task and the description of resources' capabilities. This concrete workflow has a DAG structure as it is executed by the Pegasus engine that is Condor DAGMan. Condor *Directed Acyclic Graph Manager* (DAGMan[16]) provides a workflow engine that manages Condor jobs organized as *Directed Acyclic Graphs* (DAGs). It basically allows the description of precedence constraints between Condor jobs. A weakness of this solution is that planners must operate on an entire workflow statically and execution sites can not be changed after a process is initiated. Therefore, it does not support dynamic changes. Nevertheless, intermediary results can be saved to support monitoring, history/log and reproducibility.

A successor of this system is Swift[17]. It is a system for the rapid and reliable specification, execution,and management of large-scale science and engineering workflows [86]. It supports applications that execute many tasks coupled by disk-resident datasets (as is common, for example, when analyzing large quantities of data or performing parameter studies or ensemble simulations). Swift integrates CoG Karajan, see section 6.2. It adds support for high-level abstract specification of large parallel computations, data abstraction and workflow restart.

6.6 Taverna

Taverna [65] was designed in the context of the *myGrid e-Science* UK project. It is an open-source, Grid-aware workflow management system. Taverna is implemented as a service-oriented architecture, based on Web service standards. Taverna includes a service discovery facility enabling to point at a single Web service or a site with links to multiple Web services.

The *Simple Conceptual Unified Flow Language* (SCUFL) workflow language was developed in this project [65]. It is a high level XML-based conceptual language allowing a user to specify a workflow through groups of local or remote services which are connected with data links (providing data flow) and control links (allowing coordination of services not connected through data flow). SCULF follows a data-oriented approach, provides support for processing data collections through implicit interaction mechanisms, such as the cross and dot product operators for data lists (collections).

Provenance plays an integral part in Taverna, allowing users to capture and inspect details such as who conducted the experiment, what services were used and what results were obtained. Accordingly to [40] Taverna system bind

[16]Condor DAGMan Web site at http://www.cs.wisc.edu/condor/dagman/
[17]Swift Web site at http://www.ci.uchicago.edu/swift/

workflow descriptions directly with the realizations of their building elements. In this way, it lacks any chance of dynamism.

6.7 Triana

Triana [81] is an integrated and generic workflow-based *Problem Solving Environment* and a test application for the *GridLab* project [18]. It is designed to define, process, analyze, manage, execute and monitor Grid workflows.

Triana follows both a resource oriented and a service oriented approaches. It manages two different interfaces to enable both kinds of operation: GAP (*Grid Application Prototype*) interface for service oriented tasks and GAT (*Grid Application Toolkit*) API for Grid oriented tasks. It allows the use of OGSA Grid services. Triana has been linked to the Pegasus execution planner for Frid jobs in order to create workflows to be executed on the Condor Frid system. Triana also contains JXTA based peer-to-peer module for cycle stealing that forms the basis of the project vision of a Consumer Grid, similar to the SETI@HOME project.

The workflow language is based on dataflow constructs, but it also contains many control flow operators. Loops (e.g., do, while) and execution branching (e.g., if/then) can be used to control the workflows. Triana supports multiple languages by allowing different workflow readers/writers to be plugged in, including: WSFL, DAG, BPEL and Petri net formats. The workflows can be multi-level and parts of workflow can be grouped to sub-workflows in a hierarchical way.

7 Differences in Dynamicity between Scientific and Business Workflows

Historically, business workflows have roots going back to *office automation* systems of the 1970's and 80's and gained momentum in the 90's under different names including *business process modeling* and *business process engineering*. More recently, the business workflow proposals have gained some influence in the Web services arena. For example, the Business Process Execution Language for Web Services, a merger of two earlier standards: IBM's WSFL and Microsoft's XLANG.

The question about if there are any difference between business workflows and other kinds of workflows is a recurrent one. In particular, this question has been considered by several researches [20, 51, 55, 58], identifying the following distinctions between business and scientific workflows:

- Business workflows are *task* oriented while scientific workflows are *data-flow* oriented. Scientific workflows use to involve the transportation and analysis of large quantities of data between distributed repositories.
- In business workflows documents are modified, but they still are identifiable through the process. Scientific workflows demand language constructs to manage and distribute data elements: iterations over lists (foreach), filtering, generic & higher-order operations (e.g., zip, map), etc.
- Scientific workflows involve problems with large data sets: volume, complexity, heterogeneity. Sometimes data is produced over a long time period. In these cases, it is not desirable to wait to initiate the following task until all the data has been produced.
- Scientific workflows need the involvement of large computational resources, requiring the access to Grids.
- Scientific workflows need for rich user interaction and workflow *Steering*: pause/revise/resume; select and branch.
- Scientific workflows need for persistence of intermediate products and *provenance*.

In general, the requirements of business workflows and scientific workflows are quite similar, but scientific workflows use to require the management of large data sets, the operation of a large amount of computational resources and take a long time to finish. Taking into account these issues, dynamicity has been analyzed in this report as a main requirement in scientific workflows, but it is also required in business workflows [49]. Anyway, a main difference is that business workflows are less exploratory and evolving in nature than scientific workflows [13, 36]. The support of business workflows is mainly focused towards well defined processes, working on stable data sets and executed in stable executing infrastructures. In the case of scientific workflows, as it has been shown, the processes are not always very well stablished, data elements are continuously evolving and the executing infrastructure is unestable.

[18] GridLab project Web site http://www.gridlab.org

8 Conclusions

Scientific workflows are being used to support research in many different domains. This technology has been developed to facilitate the management of scientific experiments and activities involving large amounts of data that have to be processed at several computational-intensive stages. Common scientists are not experts in the management of information and communication technologies and, therefore, they need solutions at an appropriate level of abstraction that enables the application of these technologies to scientific problems. The basics of the workflow solution consisting on the separation between the specification stage and the execution stage is a natural one. As a result, scientists can focus on their scientific problems while computer engineers can focus on the development of compuational support systems.

In this report we have analyzed dynamicity as a main requirement towards the appropriate support of scientific workflows. Obviously, scientific workflows vary in their requirements and the technologies they use depending of the disciplines involved. Anyway, we have found that it is possible to unify these requirements and come up with a set of common requirements useful for multiple disciplines. In a first scenario, we have shown how the basic nature of the scientific method is intrinsically dynamic. In other cases, the problems to be solved involve the management of data that is produced dynamically or tasks that change during performance. In addition, the computational infrastructure can also introduce variations that should also be supported. Finally, we have obtained a set of dynamicity requirements to provide a clear view of the problems that need to be solved.

References

[1] *An Introduction to Scientific Research*. McGraw-Hill Book Comp., 1952.

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

[3] Anastassia Ailamaki, Yannis E. Ioannidis, and Miron Livny. Scientific workflow management by database management. In *SSDBM '98: Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, pages 190–199, Washington, DC, USA, 1998. IEEE Computer Society.

[4] Asif Akram, David Meredith, and Rob Allan. Application of business process execution language to scientific workflows. *International Transactions on Systems Science and Applications*, 1(3):289–302, 2006.

[5] M. Allen. Do-it-yourself climate prediction. *Nature*, 401:642, oct 1999.

[6] Martin Alt, Andreas Hoheisel, Hans Werner Pohl, and Sergei Gorlatch. A grid workflow language using high-level petri nets. In Roman Wyrzykowski, Jack Dongarra, Norbert Meyer, and Jerzy Wasniewski, editors, *PPAM*, volume 3911 of *Lecture Notes in Computer Science*, pages 715–722. Springer, 2005.

[7] I. Altintas, O. Barney, Z. Cheng, T. Critchlow, B. Ludaescher, S.G. Parker, A. Shoshani, and M. Vouk. Accelerating the scientific exploration process with scientific workflows. *J. Phys. : Conf. Ser.*, 46:468–478, 2006.

[8] Patrick Amestoy, Michel Dayd, Ronan Guivarch, Christophe Hamerling, and Marc Pantel. Use of Scenarios for Generating Dynamic Execution Workflows over the Grid within the Grid-TLSE project. In Theodore Simos and Georgios Psihoyios, editors, *International E-Conference on Computer Science, Electronic Conference, 10/07/06-14/07/06*, volume 8 of *Lecture Series on Computer and Computational Science*, pages 1–11, http://www.brill.nl, septembre 2007. BRILL.

[9] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, November 2002.

[10] Adam Baker and Jano van Hemert. Scientific workflow: A survery and research directions. In *Proceedings of the Third Grid Applications and Middleware Workshop (GAMW'2007)*, Gdansk, Poland, September 9-12 2007.

[11] Jean-Pierre Banâtre, Pascal Fradet, and Yann Radenac. Programming self-organizing systems with the higher-order chemical language. *International Journal of Unconventional Computing*, 3(3):161–177, 2007.

[12] Jean-Pierre Banâtre and Daniel Le Métayer. Programming by multiset transformation. *Commun. ACM*, 36(1):98–111, 1993.

[13] Roger Barga and Dennis Gannon. *Scientific versus Business Workflows*, chapter 2, pages 9–16. In [80], 2007.

[14] Louis Bavoil, Steven P. Callahan, Carlos Eduardo Scheidegger, Huy T. Vo, Patricia Crossno, Cludio T. Silva, and Juliana Freire. Vistrails: Enabling interactive multiple-view visualizations. In *IEEE Visualization*, page 18. IEEE Computer Society, 2005.

[15] Chad Berkley, Shawn Bowers, Matthew Jones, Bertram Ludäscher, Mark Schildhauer, and Jing Tao. Incorporating semantics in scientific workflow authoring. In *SSDBM'2005: Proceedings of the 17th international conference on Scientific and statistical database management*, pages 75–78, Berkeley, CA, US, 2005. Lawrence Berkeley Laboratory.

[16] Fran Berman, Geoffrey Fox, and Anthony J. G. Hey. *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons, Inc., New York, NY, USA, 2003.

[17] M. Burns, A. Rowland, D. Rueckert, J. Hajnal, K. Leung, D. Hill, and J. Vickers. Information extraction from images (ixi): Grid services for medical imaging. In *Proceedings of the DiDaMIC Workshop (MICCAI)*, Rennes, France, 2004.

[18] Steven P. Callahan, Juliana Freire, Emanuele Santos, Carlos E. Scheidegger, Cláudio T. Silva, and Huy T. Vo. Vistrails: visualization meets data management. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 745–747, New York, NY, USA, 2006. ACM.

[19] Yves Caniou, Eddy Caron, Hlne Courtois, Benjamin Depardon, and Romain Teyssier. Cosmological simulations using grid middleware. IEEE, March 26 2007.

[20] Jinjun Chen and W.M.P. van der Aalst. On scientific workflow. *IEEE Technical Committee on Scalable Computing Newsletter*, 9(1), May 2007.

[21] D. Louis Collins, Johan Montagnat, Alex P. Zijdenbos, Alan C. Evans, and Douglas L. Arnold. Automated estimation of brain volume in multiple sclerosis with biccr. In *IPMI '01: Proceedings of the 17th International Conference on Information Processing in Medical Imaging*, pages 141–147, London, UK, 2001. Springer-Verlag.

[22] Dynamic, extensible and context-aware exception handling for workflows. In *OTM Conferences (1)*, pages 95–112, 2007.

[23] Frederica Darema. Introduction to the iccs 2007 workshop on dynamic data driven applications systems. In *International Conference on Computational Science (1)*, pages 955–962, 2007.

[24] Frederica Darema and Mario Rotea. Dynamic data-driven applications systems. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 2, New York, NY, USA, 2006. ACM.

[25] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Kent Blackburn, Albert Lazzarini, Adam Arbree, Richard Cavanaugh, and Scott Koranda. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, V1(1):25–39, March 2003.

[26] Ewa Deelman, Scott Callaghan, Edward Field, Hunter Francoeur, Robert Graves, Nitin Gupta, Vipin Gupta, Thomas H. Jordan, Carl Kesselman, Philip Maechling, John Mehringer, Gaurang Mehta, David Okaya, Karan Vahi, and Li Zhao. Managing large-scale workflow execution from resource provisioning to provenance tracking: The cybershake example. In *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, page 14, Washington, DC, USA, 2006. IEEE Computer Society.

[27] Ewa Deelman and Yolanda Gil. Managing large-scale scientific workflows in distributed environments: Experiences and challenges. In *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, page 144, Washington, DC, USA, 2006. IEEE Computer Society.

[28] K. Droegemeier, T. Baltzer, A. Wilson, M. Ramamurthy, and K. Lawrence. A new paradigm for mesoscale meteorology: Grid and web service-oriented research and education in LEAD. In *Proceedigns of the AMS Conference*.

[29] D. A. Bachelor et al. Simulation of fusion plasmas: Current status and future direction. *Plasma Science and Technology*, 2007.

[30] T. Fahringer, S. Pllana, and A. Villazon. Agwl: Abstract grid workflow language. In *Proceedings of the International Conference on Computational Science, Programming Paradigms for Grids and Metacomputing Systems*, Krakow, Poland, 2004. Springer-Verlag.

[31] T. Fahringer, J. Qin, and S. Hainzer. Specification of grid workflow applications with agwl: an abstract grid workflow language. In *CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2*, pages 676–685, Washington, DC, USA, 2005. IEEE Computer Society.

[32] Geoffrey C. Fox and Dennis Gannon. Special issue: Workflow in grid systems: Editorials. *Concurrency and Computation : Practice & Experience*, 18(10):1009–1019, 2006.

[33] Dennis Gannon, Beth Plale, Suresh Marru, Gopi Kandaswamy, Yogesh Simmhan, and Satoshi Shirasuna. *Workflows for eScience: Scientific Workflows for Grids*, chapter Dynamic, Adaptive Workflows for Mesoscale Meteorology. Springer-Verlag, 2007. In Press.

[34] Jr. George Chin, L. Ruby Leung, Karen Schuchardt, and Debbie Gracio. New paradigms in problem solving environments for scientific computing. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*, pages 39–46, New York, NY, USA, 2002. ACM.

[35] C. Germain, V. Breton, P. Clarysse, Y. Gaudeau, T. Glatard, E. Jeannot, Y. Legre, C. Loomis, I. Magnin, J. Montagnat, J.-M. Moureaux, A. Osorio, X. Pennec, and R. Texier. Grid-enabling medical image analysis. *Journal of Clinical Monitoring and Computing*, 19(4-5):339–349, December 2005.

[36] Yolanda Gil, Ewa Deelman, Mark Ellisman, Thomas Fahringer, Geoffrey Fox, Dennis Gannon, Carole Goble, Miron Livny, Luc Moreau, and Jim Myers. Examining the challenges of scientific workflows. *IEEE Computer*, 40(12):26–34, December 2007.

[37] Tristan Glatard, Johan Montagnat, Diane Lingrand, and Xavier Pennec. Flexible and efficient workflow deployement of data-intensive applications on grids with MOTEUR. *International Journal of High Performance Computing and Applications*, 2007.

[38] Tristan Glatard, Gergely Sipos, Johan Montagnat, Zoltán Farkas, and Péter Kacsuk. *Workflow Level Parametric Study Support by MOTEUR and the P-GRADE Portal*, chapter 18, pages 279–299. In [80], 2007.

[39] Daniel J. Goodman. Introduction and evaluation of martlet: a scientific workflow language for abstracted parallelisation. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 983–992, New York, NY, USA, 2007. ACM Press.

[40] Tomasz Gubala and Andreas Hoheisel. Highly dynamic workflow orchestration for scientific applications. In *CoreGRID Intergation Workshop 2006 (CIW06)*, pages 309–320. ACC CYFRONET AGH, 2006.

[41] Andreas Hoheisel. Grid workflow execution service ? dynamic and interactive execution and visualization of distributed workflows. In *Proceedings of the Cracow Grid Workshop 2006*, 2006.

[42] Andreas Hoheisel and Uwe Der. An xml-based framework for loosely coupled applications on grid environments. In *Computational Science ? ICCS 2003*, volume 2657 of *Lecture Notes in Computer Science*, pages 664–665. Springer, 2003.

[43] David Hollingsworth. The workflow reference model. Technical Report TC00-1003, The Workflow Management Coalition, Hampshire, UK, January 1995.

[44] Soonwook Hwang and Carl Kesselman. Gridworkflow: A flexible failure handling framework for the grid. *hpdc*, 00:126, 2003.

[45] A.S.Z.; Roos M.; Vasunin D.; de Laat C.; Hertzberger L.O.; Breit T.M. Inda, M.A.; Belloum. Interactive workflows in a virtual laboratory for e-bioscience: The sigwin-detector tool for gene expression analysis. *e-Science and Grid Computing, 2006. e-Science '06. Second IEEE International Conference on*, pages 19–19, Dec. 2006.

[46] Wesley M. Johnston, J. R. Paul Hanna, and Richard J. Millar. Advances in dataflow programming languages. *ACM Comput. Surv.*, 36(1):1–34, 2004.

[47] Niels Joncheere, Wim Vanderperren, and Ragnhild Van Der Straeten. Requirements for a workflow system for grid service composition. *Lecture Notes in Computer Science*, pages 365–374, 2006.

[48] Gail E. Kaiser, Stephen E. Dossick, Wenyu Jiang, Jack Jingshuang Yang, and Sonny Xi Ye. WWW-based collaboration environments with distributed tool services. *World Wide Web*, 1(1):3–25, 1998.

[49] Peter J. Kammer, Gregory Alan Bolcer, Richard N. Taylor, Arthur S. Hitomi, and Mark Bergman. Techniques for supporting dynamic and adaptive workflow. *Computer Supported Cooperative Work*, 9(3/4):269–292, 2000.

[50] Scott A. Klasky, Bertram Ludaescher, and Manish Parashar. The center for plasma edge simulation workflow requirements. In *ICDEW '06: Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW'06)*, page 73, Washington, DC, USA, 2006. IEEE Computer Society.

[51] Gregor Laszewski and Mike Hategan. Workflow concepts of the java cog kit. *Journal of Grid Computing*, 3(3-4):239–258, September 2005.

[52] Edward A. Lee, C. Hylands, J. Janneck, J. Davis II, J. Liu, X. Liu, S. Neuendorffer, S. Sachs M. Stewart, K. Vissers, and P. Whitaker. Overview of the ptolemy project. Technical Report UCB/ERL M01/11, EECS Department, University of California, Berkeley, 2001.

[53] Lifeng Liu, Dominik Meier, Mariann Polgar-Turcsanyi, Pawel Karkocha, Rohit Bakshi, and Charles R. G. Guttmann. Event-driven workflow management for medical image processing and analysis in a large image database. In *Proceedings of Distributed Databases and processing in Medical Image Computing*, pages 74–83, Rennes, France, October 2004.

[54] Bertram Ludaescher, Shawn Bowers, Timothy McPhillips, Norbert Podhorszki, and Norbert Podhorszki. Scientific workflows: More e-science mileage from cyberinfrastructure. In *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, page 145, Washington, DC, USA, 2006. IEEE Computer Society.

[55] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation : Practice & Experience*, 18(10):1039–1065, 2006.

[56] Ketan C. Maheshwari, Silvia D. Olabarriaga, Charl P. Botha, Jeroen G. Snel, Johan Alkemade, and Adam Belloum. Problem solving environment for medical image analysis. In *CBMS '07: Proceedings of the Twentieth IEEE International Symposium on Computer-Based Medical Systems*, pages 165–170, Washington, DC, USA, 2007. IEEE Computer Society.

[57] Andrew Stephen McGough, Jeremy Cohen, John Darlington, Eleftheria Katsiri, William Lee, Sofia Panagiotidi, and Yash Patel. An end-to-end workflow pipeline for large-scale grid computing. *Journal Grid Computing*, 3(3-4):259–281, 2005.

[58] Timothy M. McPhillips, Shawn Bowers, and Bertram Ludscher. Collection-oriented scientific workflows for integrating and analyzing biological data. In Ulf Leser, Felix Naumann, and Barbara A. Eckman, editors, *DILS*, volume 4075 of *Lecture Notes in Computer Science*, pages 248–263. Springer, 2006.

[59] Simon Miles, Paul Groth, Miguel Branco, and Luc Moreau. The requirements of using provenance in e-science experiments. *Journal of Grid Computing*, 5(1):1–25, 2007.

[60] Johan Montagnat. *Procesing and analyzing large medical image sets*. PhD thesis, University of Nice-Sophia Antipolis, Sophia Antipolis, France, December 2006. HDR thesis.

[61] Falk Neubauer, Andreas Hoheisel, and Joachim Geiler. Workflow-based grid applications. *Future Generation Computer Systems*, (22):6–15, 2006.

[62] Anne Hee Hiong Ngu, Nicholas Haasch, and Timothy McPhilips and. Technical report.

[63] Anne Hee Hiong Ngu, Nicholas Nicholas, Timothy McPhilips, Shawn M. Bowers, Bertram Ludaescher, and Terence Critchlow. Flexible scientific workflows using frames and dynamic embedding. Technical Report TXSTATE-CS-TR-2007-7, Texas State University, April 2007.

[64] H.; Goul M. Nichols, J.; Demirkan. Autonomic workflow execution in the grid. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 36(3):353–364, May 2006.

[65] Tom Oinn, Matthew Addis Justin, Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R. Pocock, Anil Wipat, and Peter Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics Journal.*, online, June 16, 2004.

[66] Slvia D. Olabarriaga, Jeroen G. Snel, Charl P. Botha, and Robert G. Belleman. Integrated support for medical image analysis methods: from development to clinical application. *IEEE Trans Inf Technol Biomed*, 11(1):47–57, 2007.

[67] Stphane Operto, Jean Virieux, Patrick Amestoy, Luc Giraud, and Jean-Yves L'Excellent. 3d frequency-domain finite-difference modeling of accoustic wave propagation using a masively parallel direct solver: a feasable study. page 2265?2269, October 1-6 2006.

[68] C. Pautasso and G. Alonso. Parallel computing patterns for grid workflows. In *Proc. of the 2006 Workshop on Workflows in support for Large-Scale Science (WORKS 06)*, 2006.

[69] Cesare Pautasso. *A flexible system for visual service composition*. PhD thesis, ETH Zurich, Department of Computer Science, 2004.

[70] Kassian Plankensteiner, Radu Prodan, Thomas Fahringer, Attila Kertesz, and Peter Kacsuk. Fault-tolerant behavior in state-of-the-art grid workflow management systems. Technical Report TR-0091, Coregrid, Institute on Grid Information, Resource and Workflow Monitoring Services, October 2007.

[71] Radu Prodan. Online analysis and runtime steering of dynamic workflows in the askalon grid environment. In *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 389–400, Washington, DC, USA, 2007. IEEE Computer Society.

[72] Radu Prodan and Thomas Fahringer. Dynamic scheduling of scientific workflow applications on the grid: a case study. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 687–694, New York, NY, USA, 2005. ACM.

[73] Jun Qin and Thomas Fahringer. Advanced data flow support for scientific grid workflow applications. In *International Conference on High Performance Computing, Networking Storage and Analysis (Supercomputing)*. IEEE Computer Society Press, November 2007.

[74] Jun Qin and Thomas Fahringer. Advanced Data Flow Support for Scientific Grid Workflow Applications. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC—07)*, Reno, NV, USA, November 10-16 2007. IEEE Computer Society Press.

[75] Jun Qin, Marek Wieczorek, Kassian Plankensteiner, and Thomas Fahringer. Towards a light-weight workflow engine in the askalon grid environment. In *CoreGRID European Network of Excellence Symposium*. Springer Verlag, 2007.

[76] A. Rowland, M. Burns, T. Hartkens, J. Hajnal, D. Rueckert, and D. Hill. Information extraction from images (ixi): Image processing workflows using a grid enabled image database. In *Proceedings of the DiDaMIC Workshop (MICCAI)*, Rennes, France, 2004.

[77] Asbjorn Rygg, Jiro Sumitomo, and Paul Roe. A unified model of batch and interactive scientific workflow and its implementation using windows workflow. In *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, page 36, Washington, DC, USA, 2006. IEEE Computer Society.

[78] Matthew Shields. *Control- Versus Data-Driven Workflows*, chapter 11, pages 167–173. In [80], 2007.

[79] J. G. Snel, S. D. Olabarriaga, J. Alkemade, H. Gratama van Andel, A. J. Nederveen, C. B. Majoie, G. J. den Heeten, M. van Straten, and R. G. Belleman. A distributed workflow management system for automated medical image analysis and logistics. In *CBMS '06: Proceedings of the 19th IEEE Symposium on Computer-Based Medical Systems*, pages 733–738, Washington, DC, USA, 2006. IEEE Computer Society.

[80] Ian Taylor, Ewa Deelman, Dennis Gannon, and Matthew Shields. *Workflows for e-Science*. Springer-Verlag, 2007.

[81] Ian Taylor, Ian Wang, Matthew S. Shields, and Shalil Majithia. Distributed computing with triana on the grid. *Concurrency - Practice and Experience*, 17(9):1197–1214, 2005.

[82] Gregor von Laszewski and Deepti Kodeboyina. A repository service for grid workflow components. In *ICAS-ICNS '05: Proceedings of the Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services*, page 84, Washington, DC, USA, 2005. IEEE Computer Society.

[83] Gosia Wrzesinska, Jason Maassen, and Henri E. Bal. Self-adaptive applications on the grid. In *PPoPP '07: Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 121–129, New York, NY, USA, 2007. ACM.

[84] Yonghong Yan, Barbara Chapman, , and Babu Sundaram. Air quality forecasting on campus grid environment. In *Workshop on Grid Applications: from Early Adopters to Mainstream Users, GGF14*.

[85] Jia Yu and Rajkumar Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Rec.*, 34(3):44–49, September 2005.

[86] Yong Zhao, Mihael Hategan, Ben Clifford, Ian Foster, Gregor von Laszewski, Ioan Raicu, Tiberiu Stef-Praun, and Mike Wilde. Swift: Fast, reliable, loosely coupled parallel computation. In *Proceedings of the IEEE International Workshop on Scientific Workflows*, Salt Lake City, Utah, U.S.A., 2007.

[87] Zhiming Zhao, Adam Belloum, Hakan Yakali, Peter Sloot, and Bob Hertzberger. Dynamic workflow in a grid enabled problem solving environment. In *CIT '05: Proceedings of the The Fifth International Conference on Computer and Information Technology*, pages 339–345, Washington, DC, USA, 2005. IEEE Computer Society.