Combining Local and Grid Resources in Scientific Workflows (for Bioinformatics) *

Ann-Charlotte Berglund[1], Erik Elmroth[2], Francisco Hernández[2], Björn Sandman[2], and Johan Tordsson[2]

[1] The Linnaeus Centre for Bioinformatics, Uppsala University
ann-charlotte.sonnhammer@lcb.uu.se, www.lcb.uu.se
[2] Department of Computing Science and HPC2N, Umeå University
{elmroth, hernandf, c01bsn, tordsson}@cs.umu.se, www.gird.se

Abstract. We examine some issues that arise when using both local and Grid resources in scientific workflows. Our previous work addresses and illustrates the benefits of a light-weight and generic workflow engine that manages and optimizes Grid resource usage. Extending on this effort, we here illustrate how a client tool for bioinformatics applications employs the engine to interface with Grid resources. We also explore how to define data flows that transparently integrates local and Grid subworkflows. In addition, the benefits of parameter sweep workflows are examined and a means for describing this type of workflows in an abstract and concise manner is introduced. Finally, the above mechanisms are employed to perform an orthology detection analysis.

Key words: Grid, workflow, data flow, parameter sweep, bioinformatics

1 Motivation and Background

To date many available tools (e.g., [1–3]) for scientific workflows have sophisticated functionality that enables, e.g., design, enactment, and scheduling of workflows, while providing support for data management and fault tolerance. However, most of these tools have at least one of three shortcomings: (1) the tool is a monolithic application where the above listed functionality cannot easily be extracted and reused, (2) the tool targets a specific scientific domain and to adapt it to new domains is a major, if not impossible, endeavor, and (3) the tool is focused on workflow execution on client machines ignoring the benefits of using Grid resources for computationally or data intensive tasks.

In previous work [4, 5] we have discussed these problems and argued in favor of a loosely coupled design where workflow capabilities are exposed as a composable set of workflow services with clear separation of concerns. By orchestrating these services, environments tailored to the needs of a certain group of users

* This research was conducted using the resources of the High Performance Computing Center North (HPC2N). Financial support has been provided by The Swedish Research Council (VR) under contract 621-2005-3667.

can be constructed with much less effort compared to a (re)implementation of a full-featured system. Our design and implementation of the Grid Workflow Enactment Engine (GWEE) [4] demonstrates the viability of this approach.

This work is a direct continuation of these efforts. We identify the following five contributions, each described in a separate section of the paper. First, we illustrate the feasibility of the concept of a composable set of workflow capabilities by demonstrating how a client tool, adapted for use within bioinformatics, can be built on top GWEE. The problems of decoupling data from task specification and that of dynamically adjusting workflow tasks during runtime are both addressed with task templates that delay the binding of task parameters until workflow execution. We then examine data flows integrating the client machine and Grid resources. We also present a mechanism to simplify parameter sweep studies. Finally, these results are used to reduce the complexity of a particular bioinformatics problem - orthology detection analysis.

2 GWEE Details and the Workflow Client Tool

Before discussing the data management issues encountered when combining Grid and local workflows, we give a brief overview of the GWEE and the client tool. The GWEE [4] is a Grid workflow enactment engine that is exposed as a Web service. A strict separation between workflow enactment logic and execution of individual tasks (e.g., computational jobs and file transfers) enables independence of workflow description language and interoperability with multiple Grid middlewares. The GWEE uses a data flow model of computation in which task dependencies define the execution order of the workflow. However, workflows driven (in part or in full) by a control flow are also supported.

While the GWEE provide access to Grid resources, the interaction with users is delegated to a client tool that uses the service interface to start and terminate workflows, as well as pause and resume a running workflow. In this tool workflows can also be designed, in a drag and drop manner, by connecting tasks together in a graph that specifies a control and/or a data flow. The client can handle mixed local and Grid workflows. Simple tasks that are not computationally intensive, e.g., preparation of input files, are typically performed on the client side whereas long running jobs preferably should be combined into a subworkflow that can be sent to the GWEE for enactment on the Grid. The state of (running) workflows can be stored to, and loaded from disk. The client can hence be disconnected from the GWEE during the execution of a long Grid subworkflow. This is useful, e.g., when running the client on a laptop during travel or for Grid workflows executing for days or weeks. The client tool can be updated of Grid subworkflow progress either by synchronous requests or by notifications.

In both the GWEE and the client tool, workflow tasks consist of three parts (1) a set of input ports, (2) a set of output ports, and (3) a process that maps the input ports to the output ports (i.e., a job that receives inputs and produces outputs). Links from output ports to input ports define task dependencies. Similarly to other data flow approaches [6], the links are associated with tokens

that are moved between tasks. For the GWEE these tokens represent pointers to data files with the transfer medium dependent upon the supported infrastructure (e.g., GridFTP) and the transfer carried out by middleware specific plugins. On the client side, a token holds the name of a locally stored file that can be accessed from both tasks in a producer/consumer fashion.

3 Task templates

Scientific workflows running on the Grid typically consist of self-contained command line applications or scripts. Execution is usually configured via environment variables or command line arguments and all communication, in and out, is performed via data files or the standard streams (i.e., *stdin*, *stdout*, and *stderr*). While this task configuration information must be explicitly described prior to submission to Grid resources, it is not necessary when designing the workflows. The binding of particular parameters can be delayed until workflow enactment, which is beneficial since the required information is not always known at design time. In addition, to support non-deterministic[3] workflows, parts of the workflow description must be completed during enactment with values produced as output from (previously) executed workflow tasks.

The use of command line applications introduces the problem of embedding the required task information into the workflow model. For example, as mentioned in the previous section, the GWEE follows a data flow approach in which all information is moved through ports. Values passed through these ports (e.g., name of input files) are required to enact the workflow and correspond to the arguments and data products of the command line applications. This introduces the difficulty of mapping the arguments to the corresponding input and output ports for the correct arrangement of dependencies delineating the flow of data.

An often employed solution is to use wrappers to either include individual applications into the workflow model or to execute those applications in a Grid [7–9]. GEMLCA [7] exposes a generic service layer to execute command line applications without the need of re-engineering the application. The Styx Grid Services [8] create services of command-line applications and allows them to run remotely as if they were local programs. In Styx, shell scripts can be used to compose the services as workflows. These solutions enable the reuse of the applications with different data [10] but require a service for each individual application. If services are used only for a short period of time, these solutions become too heavyweight as the overhead of service creation outweighs the execution time of the wrapped application. In non-Grid environments, workflow tools [1–3] usually offer a framework with a set of functionalities that can be extended by writing framework specific applications using libraries developed exclusively for this purpose [11]. However, when moved to the Grid this solution is not always feasible as the application source code need not be accessible.

[3] By the term *non-deterministic*, we refer to a workflow where the precise enactment is not known in advance and it is determined dynamically during execution. *Dynamic* workflows, in contrast, are actively steered by users.

Compared to the wrapper solution, our approach is lightweight as it involves only the specification of a parameterized *task-template* where the workflow designer specifies the correct mapping of input and output ports to the parameters of the application. The designer can also specify which arguments are to be determined at a later stage (during enactment). A task-template is an incomplete task description, in JSDL [12], that contains name-value mappings for missing task parameters. The template is transformed into an executable task once the values for these missing parameters are available. The actual values for the parameters are defined (and updated) using the task ports.

By filling out a single form that is automatically generated from the task template, users can specify values for the template parameters without being concerned about low-level details such as the syntax of the workflow and/or task descriptions. The template functionality also allows the transformation of the task outline to the resulting task to be delayed until task execution time, and hence enables non-deterministic workflows.

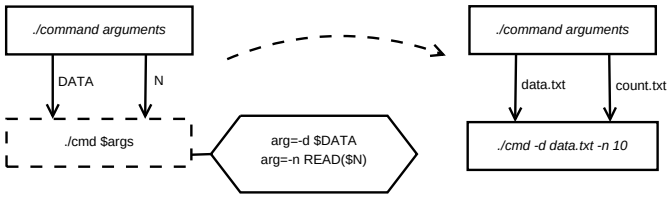

Fig. 1. Basic operation of the template, including usage of the read operation.

For some task templates, the value specified in the input port is not sufficient information for the workflow to be enacted correctly. Instead, the data value(s) produced at that port is required. To access these values, we provide a *read* operation that uses the port content instead of the port name for task generation in the template. The POSIX command *xargs* provides a functionality similar to the read task, but is restricted to mapping stdin to command line arguments.

A simplified illustration of the template functionality, including the read operation, is shown in Figure 1. In this figure, the incomplete task description (dashed box) contains a template (shown as a hexagon). After receiving values (data.txt, count.txt) for the template parameters ($FILE, $N), the task outline is transformed to the executable task, shown to the right. Note how the read operation enables the content (10) of the port marked N to be used instead of the name of the token passed on that port (count.txt).

4 Transparent Local and Grid Data Flows

Data transfers become cumbersome for data flow workflows that contain both local and Grid subworkflows. Special care is required to ensure that the output of a local task is available as input to a subsequent Grid task. To handle this, the

workflow client explicitly transfers the data to a Grid-available storage element and then back to the client machine once the Grid subworkflow is completed. These transfers, being control flow activities, are hidden from the user and the visual workflow displayed in the client tool remains pure data flow.

Input ports can be used not only to define input from previous tasks in the workflow, but also to specify dependencies on external data. In Grid subworkflows, this feature is utilized to automatically transfer required files, e.g., binaries and other task input files not originally produced by the workflow.

5 Parameter Sweeps in Workflows

A commonly used pattern in workflows is parameter sweeps, where one computational task (or a subworkflow) is executed with different data. Parameter sweep require a suitable partition of the input data where each partition is assigned to a different task (or subworkflow). Embarrassingly parallel problems such as parameter sweep studies are well suited to Grid environments, where large numbers of resources are available. Even though parameter sweep workflows can be done without Grid workflows, data management in the client is simplified when the parameter sweep can be treated as one (Grid) subworkflow, instead of a (potentially large) set of individual tasks. This increased abstraction is beneficial both from a usability and a performance point of view. Other advantages include the reuse of a familiar programming paradigm akin to map-reduce or scatter-gather reductions. By implementing parameter sweeps as a workflow graph rewrite on the client side, iterations of the *parallel-for* style are possible, regardless of whether or not iterations are supported by the workflow engine.

In data flow languages, parameter sweeps are typically implemented using either graph rewrites, see e.g., Triana [3] and Pegasus [13] or with higher-order functions as done, e.g., by Kepler [1]. Generation of the vector to sweep over can be done statically before the execution as in Triana [14], or dynamically during workflow enactment, the latter approach used by Askalon [15], Kepler [1], and Taverna [2]. The P-GRADE [10] data composition strategies allow a custom pairing of service input ports, typically through dot or cross product. Pautasso et al [16] study parameter sweeps and other parallel patterns for workflows.

Our implementation of parameter sweeps is based on a graph rewrite that can take place either before or during workflow execution. A parameter sweep consists of three parts: i) a *divider* that generates the desired partition, ii) the task template(s) forming the subworkflow that should be swept over, and iii) a *merger* that collects the results. The divider and the corresponding merger are computation tasks in the workflow. The rationale behind this is that the workflow designer knows best how to properly generate the parameter sweep instances, i.e., the branches in the parameter sweep. By allowing the designer to implement the merger and divider (typically done in a Turing-complete scripting language) rather than to choose from a few predefined partition types, new parameter distribution patterns can be added without modifying the workflow language. Commonly used patterns include for each input file, for each part of a large

file (that is to be split up), and Cartesian product between two sets, the latter illustrated in the bioinformatics use case in Section 6.

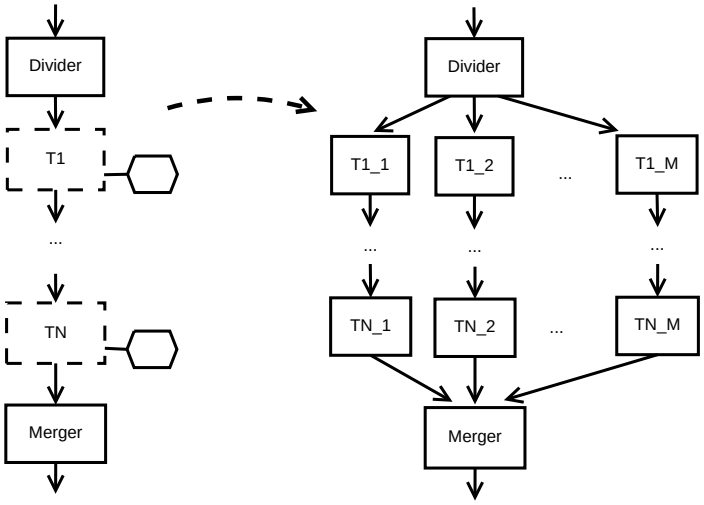

Fig. 2. Expansion of a subworkflow into a parameter sweep.

The divider task output generates the desired data partition for the parameter sweep. Each line of output contains template value bindings for one branch in the parameter sweep. When the divider task completes, the template subworkflow is rewritten (expanded) into a task subworkflow, that can be executed either locally or on the Grid. The divider task can either be executed statically before the workflow is started or dynamically during workflow enactment. In the latter case, the divider itself can contain templates that allow information collected during workflow execution guide the generation of sweep instances. The merger is always executed during workflow run time. It can be configured either to be part of the generated (Grid) subworkflow or be executed locally.

An illustration of the parameter sweep expansion process is shown in Figure 2. From the M lines of divider task output, the incomplete task descriptions (dashed boxes T1 ... TN) with their corresponding templates (depicted as hexagons) are resolved into sweep instances, each with concrete tasks. The parameter sweep in Figure 2 is static as the divider itself does not make use of the template functionality to determine M during workflow enactment. For clarity, the figure shows a set of tasks connected as a pipeline, but the parameter sweep functionality can be applied to any connected subworkflow.

6 InParanoid - a Bioinformatics Use Case

With the new sequencing technologies [17], the bioinformatics field is facing an avalanche of sequence data to be analyzed. As many of the computational problems are embarrassingly parallel, there is a growing interest in the bioinformatics field for Grid techniques in order to accelerate the data processing.

The analysis of biological data is typically a mix of short processing steps and larger computations that can easily be parallelized. These steps are all part of a scientific discovery workflow for answering a biological question. Therefore, for a workflow tool to be useful in the bioinformatics field it must have the ability to combine local and Grid subworkflows into a larger workflow. The large-scale bioinformatics analyzes are made on general purpose computational Grids, such as EGEE and NorduGrid, and can hence not rely on pre-installed software. All this considered, these analyzes are today not done using a workflow tool. Instead, the different tasks in the analysis are typically connected by Perl scripts and are manually submitted to the computational Grids. The drawbacks with this approach are that (1) it can be difficult to exactly replicate the analysis, (2) there is no easy way to reuse the scientific discovery workflow, and (3) it is difficult to analyze and verify this workflow.

Here we have implemented a workflow for the orthology detection tool InParanoid [18]. Orthology and paralogy are key concepts in comparative genomics, and both refer to genes that are related through common descent, i.e., genes that are homologous. In the former case through speciation and in the latter through duplication. Due to the different evolutionary relationships, orthologous genes are more likely to have preserved the biological role than paralogous genes. Therefore, it is important to be able to distinguish between these two types of homologous genes during, for example, the functional annotation of newly sequenced genomes. The data used here are the collected protein sequences, the so-called proteomes, from five species[4] Candida glabrata (NCBI), Drosophila pseudoobscura (FlyBase), Escherichia coli K12 (NCBI), Kluyveromyces lactis (NCBI), and Saccharomyces cerevisiae (SGD).

The InParanoid workflow consists of four steps: (1) format gene databases and filter the proteomes to get the longest transcript per gene, (2) run the Basic Local Alignment Search Tool (BLAST) [19] on all pairs of proteomes including self-against-self, (3) filter the BLAST results, and (4) run the InParanoid application. These four steps are illustrated in Figure 3. Step 1 is a foreach-style parameter sweep that executes locally. The Grid executed parameter sweep in Step 2 performs sequence similarity searches using the BLAST tool. Step 3 is a local parameter sweep and Step 4 is a Grid subworkflow. The parameter sweeps in steps 2 and 3 generate sweep instances according to a Cartesian product pattern. The InParanoid workflow makes extensive use of the template functionality, e.g., to pass genome file names and gene transcript lengths to the command line applications. The edges in the resulting workflow graph in Figure 3 depict a simplified control flow. The complete workflow combines local control flow and Grid data flow, the latter used to install the binaries (BLAST and InParanoid) and to transfer the gene databases and results.

The InParanoid workflow was run on a dedicated Grid testbed with four resources that run Ubuntu Linux 2.6.24, NorduGrid/ARC 0.6, Maui 3.2.6, and Torque 2.1.9. Each resource had one HP DL165 G5 Opteron 2346 HE, with a

[4] NCBI, FlyBase, and SGD refer to the respective databases where the proteome data are publicly available.

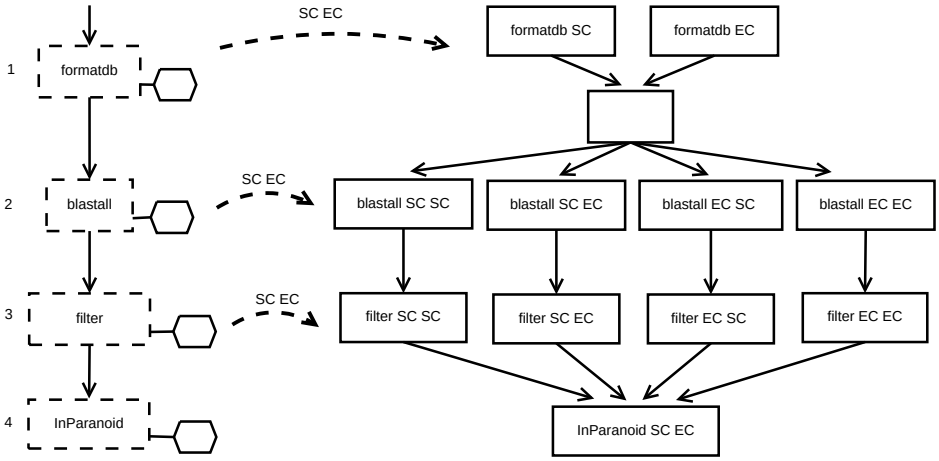

Fig. 3. A simplified view of the templates (left) and the resulting InParanoid workflow (right) instantiated for two proteomes (SC and EC). The numbers $1, \ldots, 4$ correspond to the four steps described in Section 6.

quad core 1.8 GHz CPU and 4GB of memory as backend. The local steps in the workflow were enacted on Ubuntu Linux 2.6.27 laptop with an Intel 1.2GHz Core 2 Duo CPU and 4 GB memory. A 100 Mbit/s network connected all machines.

Table 1 shows performance results for pairwise InParanoid comparisons of the five investigated proteomes: Drosophila pseudoobscura (DP) with 9871 protein sequences, Saccharomyces cerevisiae (SC) with 5792 sequences, Kluyveromyces lactis (KL) with 5336 sequences, Candida glabrata (CG) with 5192 sequences, and Escherichia coli K12 (EC) with 4243 sequences.

Notably, as exactly the same workflow is enacted when using one and four Grid resources and Step 2 is the only parallel part, the two runs give similar results for all other columns in Table 1. The runtime complexity of BLAST is $\mathcal{O}(mn)$, where m and n are the sizes of the compared proteoms. The obtained speedup hence varies with the proteom size induced load (im)balance of Step 2. Notably, the varying from one to four resources only applies to the Grid workflow in Step 2, although the speedup is calculated for the whole workflow. The last column of Table 1 shows that the parallel part (Step 2) constitutes between 85 and 95 per cent of the total execution time. We remark that 90 per cent parallel execution time would, with perfect load balance and neglectable overhead, result in a speedup of 3.08 for four resources. In traditional parallel applications that execute in a cluster and communicate through message passing, poor speedup due to load imbalance results in idle nodes and wasted CPU cycles. On the contrary, in the InParanoid workflow, there is no waste of resources, as cluster nodes can be used by other tasks once the shorter BLAST jobs in Step 2 completes. Notably, the parallel speedup of the InParanoid workflow could be improved with a finer-grained data distribution pattern for Step 2, e.g., by partitioning the proteom files into equally sized chunks. We however remark that the above

Table 1. Performance results for the InParanoid workflow for proteom pairs as given in the first column. The columns labeled 1 to 4 shows the time for workflow steps 1 to 4. Next follows the overhead (which is also part of the time in the column labeled 2) for file transfer to and from the Grid, all other workflow enactment overhead, the makespans of the complete workflow enacted with four and one Grid resources, and the associated speedup (calculated as MS(1)/MS(4)), respectively. The last column shows how large part Step 2 (the part to be executed in parallel) constitutes of the makespan for the one resource case. All time units are in seconds.

Proteoms	1	2	3	4	File staging	OH	MS(4)	MS(1)	Speedup	T_p(%)
DP-SC	2	5639	340	737	152 (1581MB)	19	6737	14213	2.11	92.2
DP-KL	2	5606	297	799	134 (1381MB)	17	6720	12894	1.92	91.5
DP-CG	2	5695	321	768	147 (1503MB)	20	6806	13590	2.00	92.0
DP-EC	1	5581	189	219	92 (899MB)	15	6005	8408	1.40	94.9
SC-KL	1	2256	188	814	94 (922MB)	19	3278	8098	2.47	87.4
SC-CG	1	2226	226	1043	110 (1096MB)	18	3514	9304	2.65	85.9
SC-EC	1	2233	85	219	53 (436MB)	17	2555	4044	1.58	93.3
KL-CG	1	1948	172	768	89 (844MB)	18	2907	7599	2.61	87.2
KL-EC	1	1567	58	189	41 (307MB)	15	1830	3258	1.78	91.5
CG-EC	1	1891	74	204	47 (383MB)	16	2186	3647	1.67	92.4

tests are an illustration of the functionality of the parameter sweep mechanism, rather than an attempt to optimize the performance of the InParanoid workflow.

7 Conclusions and Future Work

We demonstrate how data flows seemlessly can integrate local and Grid resources. We also introduce a more flexible parameter sweep tool than those available in current workflow systems. Our task template mechanism enables non-deterministic workflows and delineates data flow for command line applications. Through a combination of these mechanisms, we illustrate how to simplify incorporation of Grid resources in the InParanoid workflow. Future work includes analyzing conceptual interoperability aspects for scientific workflows to lay a foundation for workflow reuse and workflow system interoperability.

References

1. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Leen, E.A., Tao, J., Zhao, Y.: Scientific workflow management and the Kepler system. Concurrency Computat.: Pract. Exper. **18**(10) (2006) 1039–1065
2. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M., Wipat, A., Li, P.: Taverna: A tool for the composition and enactment of bioinformatics workflows. Bioinformatics **20**(17) (2004) 3045–3054
3. Taylor, I., Shields, M., Wang, I., Harrison, A.: The Triana workflow environment: architecture and applications. In Taylor, I., et al., eds.: Workflows for e-Science. Springer (2007) 320–339

4. Elmroth, E., Hernández, F., Tordsson, J.: A light-weight Grid workflow execution engine enabling client and middleware independence. In Wyrzykowski, R., et al., eds.: Parallel Processing and Applied Mathematics, LNCS 4967, Springer 754–761
5. Elmroth, E., Hernández, F., Tordsson, J., Östberg, P.O.: Designing service-based resource management tools for a healthy Grid ecosystem. In Wyrzykowski, R., et al., eds.: Parallel Processing and Applied Mathematics, LNCS 4967, Springer-Verlag 259–270
6. Johnston, W.M., Hanna, J.R.P., Millar, R.J.: Advances in dataflow programming languages. ACM Computer Surveys **36**(1) (2004) 1–34
7. Delaitre, T., Kiss, T., Goyeneche, A., Terstyanszky, G., Winter, S., Kacsuk, P.: GEMLCA: Running legacy code applications as grid services. J. Grid Computing **3**(1–2) (2005) 75–90
8. Blower, J.D., Harrison, A.B., Haines, K.: Styx Grid Services: Lightweight middleware for efficient scientific workflows. Scientific Programming **14**(3–4) (2006) 209–216
9. Wang, I., Taylor, I., Goodale, T., Harrison, A., Shields, M.: gridMonSteer: Generic Architecture for Monitoring and Steering Legacy Applications in Grid Environments. In Cox, S., ed.: The UK e-Science All Hands Meeting 2006. (2006) 769–776
10. Glatard, T., Sipos, G., Montagnat, J., Farkas, Z., Kacsuk, P.: Workflow-level parametric study support by MOTEUR and the P-GRADE portal. In Taylor, I., et al., eds.: Workflows for e-Science. Springer (2007) 279–299
11. Huang, Y., Taylor, I., Walker, D.W., Davies, R.: Wrapping legacy codes for grid-based applications. In: International Parallel and Distributed Processing Symposium (IPDPS'03), IEEE Computer Society (2003) 139–145
12. Anjomshoaa, A., Brisard, F., Drescher, M., Fellows, D., Ly, A., McGough, A.S., Pulsipher, D., Savva, A.: Job Submission Description Language (JSDL) specification, version 1.0 http://www.ogf.org/documents/GFD.56.pdf, February 2009.
13. Gil, Y., Ratnakar, V., Deelman, E., Mehta, G., Kim, J.: Wings for Pegasus: Creating large-scale scientific applications using semantic representations of computational workflows. In Cheetham, W., Goker, M., eds.: 19th Conference on Innovative Applications of Artificial Intelligence (IAAI). (2007) 1767–1774
14. Churches, D., Gombas, G., Harrison, A., Maassen, J., Robinson, C., Shields, M., Taylor, I., Wang, I.: Programming scientific and distributed workflow with Triana services. Concurrency Computat.: Pract. Exper. **18**(10) (2006) 1021–1037
15. Fahringer, T., Prodan, R., R.Duan, Hofer, J., Nadeem, F., Nerieri, F., Podlipnig, S., Qin, J., Siddiqui, M., Truong, H.L., Villazon, A., Wieczorek, M.: ASKALON: A development and Grid computing environment for scientific workflows. In Taylor, I., et al., eds.: Workflows for e-Science. Springer (2007) 450–471
16. Pautasso, C., Alonso, G.: Parallel computing patterns for grid workflows. In: Proc. of the HPDC2006 Workshop on Workflows in Support of Large-Scale Science (WORKS06) Paris France. (June 2006)
17. Hall, N.: Advanced sequencing technologies and their wider impact in microbiology. J. Exp. Biol. **9**(210) (2007) 1518–1525
18. Remm, M., Storm, C.E.V., Sonnhammer, E.L.L.: Automatic clustering of orthologs and in-paralogs from pairwise species comparisons. J. Mol. Biol. **314** (2001) 1041–1052
19. Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J.: Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res. **17**(25) (1997) 3389–3402