Workflows for Intelligent Monitoring using Proxy Services

Stefan RÜPING[a,1], Dennis WEGENER[a], Stelios SFAKIANAKIS[b] and Thierry SENGSTAG[c]
[a] *Fraunhofer IAIS, Schloss Birlinghoven, 53754 St. Augustin, Germany*
[b] *Biomedical Informatics Laboratory, Institute of Computer Science FORTH, Greece*
[c] *Swiss Institute of Bioinformatics, Bâtiment Génopod, 1015 Lausanne, Switzerland*

Abstract. Grid technologies have proven to be very successful in the area of eScience, and in particular in healthcare applications, where they allow combining flexible, high-speed data processing with essential requirements of data security and privacy. While the applicability of workflow enacting tools for biomedical research has long since been proven, the practical adoption into regular clinical research has some additional challenges. In this paper, we investigate the case of data monitoring, and how to seamlessly implement the step between a one-time proof-of-concept workflow and high-performance on-line monitoring of data streams, as exemplified by the case of long-running clinical trials. We will present an approach based on proxy services that allows executing single-run workflows repeatedly with little overhead.

Keywords. Grid, Workflow, Monitoring, Proxy Services

1. Introduction

Grid technologies [1] have proven to be very successful in the area of eScience where they allow to combine flexible, high-speed and large-volume data processing with essential practical considerations such as cross-organizational security and access control, resource management and load balancing, billing, or data security and privacy. In particular the latter is of essential importance in the field of healthcare in general and clinical trials in particular, which we will investigate in this paper. Several legal and ethical requirements and guidelines exist which have to be taken into account in the setup of an eHealth Grid system [2].

The applicability of workflow enacting tools for biomedical research has long since been proven [3]. However, the practical adoption into regular clinical research has some additional challenges. In this paper, we investigate the case of long-running clinical trials and how to seamlessly implement the step between a one-time proof-of-concept workflow and high-performance on-line monitoring of data streams

A clinical trial is a formal research procedure in which a novel treatment (or therapeutic procedure) is compared to an existing treatment to assess its efficiency. The questions addressed by the trial can be related to drug toxicity, severity of side effects, improvement of patient survival, etc. As part of the trial planning, a statistical design is

[1] Corresponding Author (stefan.rueping@iais.fraunhofer.de).

thus setup, in which the number of patients expected in each alternative treatment branch is defined, based on the anticipated difference in treatment outcome. After the beginning of the trial, patients are gradually enrolled and a continuous monitoring of treatment outcome is performed. The trial is stopped in the following cases:

1. Enough data has been collected to satisfactorily answer the scientific questions as specified in the definition of the trial.
2. Convincing evidence exists that one of the alternative treatments is significantly worse than the alternative. This may either be the case because the new treatment exhibits some severe side effects, or because it has been proven to be significantly more effective than the existing alternatives. In this case, it would be unethical to treat any more patients with the less effective alternative, and the trial has to be closed.

While the first case is straight-forward to implement with current tools, because it only requires a one-time data analysis after the data collection phase has been finished, the second case actually requires an event-driven continuous monitoring of the data of all patients in the trial. When we consider the case of an end-to-end system, new data may arrive at any time, driven by the researchers involved in the trial may introduce a severe delay in recognizing safety hazards to the patients, possibly violating ethical requirements. Instead, some kind of automatic alerts will be necessary. The straight-forward solution to this problem would be to repeatedly execute the data analysis workflow in short time intervals. However, considering the potential of an online integration of clinical trials with electronic healthcare records (i.e. faster access to higher volumes of data per patient), and the fact that an organization typically executes multiple trials in parallel (with numbers expected to increase with the availability of integrated clinical trial systems), it is clear that this solution does not scale well enough to be practical.

The work in this paper is based on the ACGT[2] project [4], which has the goal of implementing a secure, semantically enhanced end-to-end system in support of large multi-centric clinico-genomic trials, meaning that it strives to integrate all steps from the collection and management of various kinds of data in a trial up to the statistical analysis by the researcher. From the technological point of view, ACGT offers a modular environment in which new data processing and data mining services can be integrated as plug-ins as they become available. ACGT also provides a framework for semantic integration of data sources (e.g., clinical databases) and data mining tools, through the use of a specifically developed ontology and of a semantic mediator. In the current version, the various elements of the data mining environment can be integrated into complex analysis pipelines through the ACGT workflow editor and enactor.

In this paper, we will present an approach based on so-called proxy services that allows to easily convert single-run workflows into data monitoring tools with little overhead. The remainder of the paper is structured as follows: first, we will give an introduction to proxy services in general, before we describe how to use them to convert regular workflows into monitoring services. An experimental evaluation on a real-world biomedical data analysis workflow follows. We close the paper with some outlook into future work.

[2] http://eu-acgt.org

2. Proxy Services

In ACGT a generic data protection framework has been defined which is based on a technical security infrastructure as well as on organizational measures and contractual obligations [1]. The definition of such a framework was required so that sensitive patient data are managed in a secure, authorized, and audited way. Most of these security requirements are dealt with the Grid infrastructure layer. In particular the Grid Security Infrastructure (GSI) [5] supports user authentication through digital signatures and also the delegation of user privileges to a service so that this service performs an action on the user's behalf and without the user's intervention. The delegation mechanism is important because it allows "single sign on" for the users of the Grid.

On the other end of the spectrum we have chosen BPEL as the underlying workflow technology, which is the de facto standard for web Services orchestration and business process modeling [6]. Nevertheless the choice of BPEL requires an infrastructure that would make possible the invocation of the ACGT secure grid services from inside the BPEL-based workflows. Especially the use of GSI for securing Web Services requires a BPEL Engine ("enactor") that is able to invoke them without compromising security. In particular, each BPEL workflow should also be a GSI secured service that is able to accept the delegated user credentials and subsequently delegate them further to all the services that need to be contacted in the context of the specific workflow. Unfortunately BPEL and the Web Services standard security specifications do not deal with such requirements. This problem required the introduction of "Proxy" services that offer the bridge between the business process view of BPEL engines and the Grid secure services of the ACGT ecosystem.

The architecture of the proxy services mechanism is depicted in Figure 1. Each

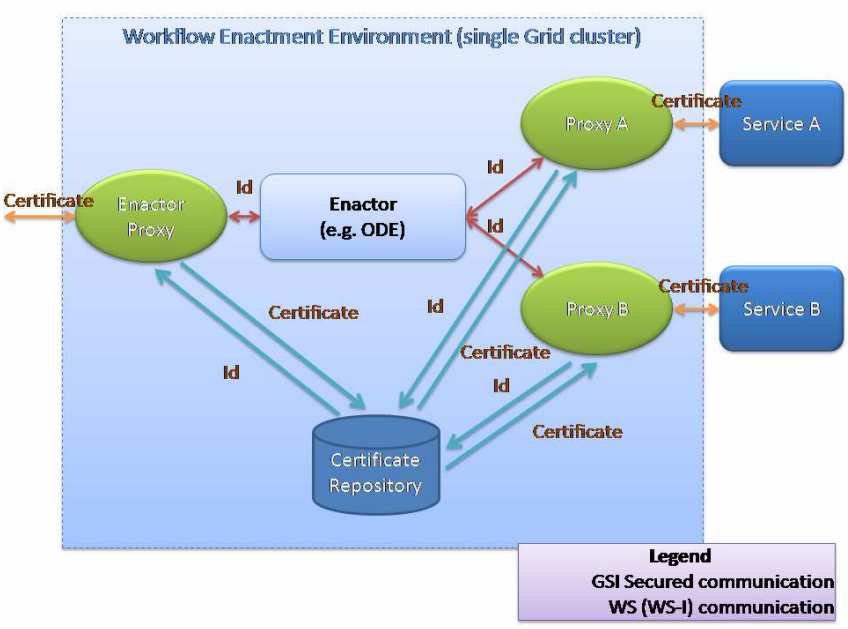

Figure 1. WorkflowEnactment Environment.

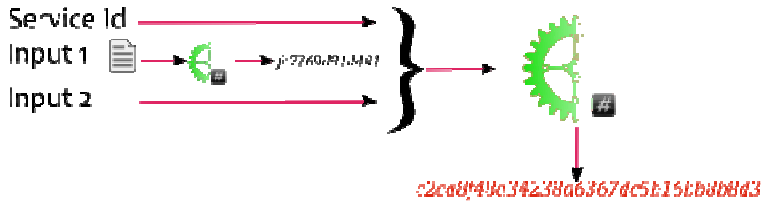

Figure 2. The generation of a unique signature of the invocation parameters.

"real" ACGT service is mirrored by a corresponding "proxy service". The proxy service has the same programmatic interface with the original service and, in its implementation, forwards all requests to the service that it mirrors. The most important difference between the proxy service and the target service is that the proxy service is not contacted through GSI but through “standard” Web Services technologies.

The introduction of the proxies is to make the BPEL engine agnostic of the whole GRID security framework. In order to achieve this, the BPEL workflows that are deployed in the enactor do not contain the invocation of the original ACGT services but the invocation of the corresponding proxy services in their places. The proxy services should be able to make GSI-secured invocations to their real parties and therefore they need access to the user's certificate that originated the workflow execution. To do that the user's credentials are “intercepted” in a service just before the enactor that we call "enactor proxy". The enactor proxy accepts the request for the execution of the workflow as if it was the real enactor, extracts the security information, creates a new proxy certificate with delegation, and stores this certificate in a certificate repository under a newly created “enactment” ID. It subsequently makes the normal call to the real enactor passing, in addition to the workflow input parameters, the ID of the certificate.

The final piece to make the whole thing work is the BPEL workflows: they are constructed in such a way so that the enactment ID that references the certificate in the database is passed as an additional parameter to all the proxy services contacted throughout the execution of the workflow. The proxy services use this ID to retrieve the stored user credentials and make GSI-secured invocation to the real ACGT service.

In order to support the repeated execution of workflows in an efficient way it is necessary to take advantage of the results of past executions. This is a well known programming technique termed “memoization”, in the sense that the previous results are “remembered” and reused, wherever this is possible, to make future executions go faster. In our case the repeated executions of the workflows that are monitored can be speeded up if many of the participating data processing steps need not be performed because their inputs have not be changed. There is a strong requirement in order for this optimization to take place though: these processing steps should be deterministic or “referentially transparent”, which means that for the same set of inputs they always yield the same outputs. Even if it appears to be a strong requirement, a lot of analytical and processing activities are indeed of this kind. Counterexamples include external databases that may be updated independently.

In order for the proxies to work in a workflow monitoring scenario they need first of all to be aware if the target service is deterministic. This is part of the target service’s metadata and therefore it is readily available by the ACGT service registry when given the specific service id. The second requirement is that the proxy services need to be aware if they are called in the context of a workflow monitoring task. This is

implemented through the incorporation of a “monitoring context” in the information that is available through the enactment ID that they accept in all of their requests. A final requirement is for the proxy services to store in this “monitoring context” past results of their invocation along with the corresponding input values.

A final important note to make is how inputs that are supplied by reference should be handled. An example of such a case is when an input parameter is id of a file stored in the Grid data management service (DMS). In general, subsequent invocations with the same input reference cannot be handled as identical because the contents of the referenced entity may have changed in between. In this case some “deep” checking of inputs equality should be performed such that the decision is not based on the reference but on the value (e.g. file contents) that is referenced. Nevertheless, in the case of the Grid DMS and its use in the context of the ACGT, equality of the file references means equivalent contents because files are not updated: new files, and therefore file ids, are created instead.

Storing the complete input values can result in a substantial cache size for large inputs. A possible alternative is the use of “*message digest*” algorithms that accept an arbitrary size of data and perform a hash function to produce a small string of characters (the “hash value” or “fingerprint”) of their input. Examples of these algorithms are the MD5, and the newer SHA family of hash functions. The most important feature of these algorithms is that it is extremely rare for two different inputs to have the same output digest, meaning that a significant reduction in cache size is traded against an extremely small chance of caching errors. Furthermore in order to deal with the situation that the same data have been processed using different grid file references, the proxy service implementation, after every invocation of the target service, “de-references” the parameters, i.e. downloads the files, to compute the hash value of the data themselves that it then incorporates in the XML document to produce the final hash value. This double hashing process is shown in Figure 2 above, where the “Input 1” is a file identifier. In conclusion, past results are cached and indexed both by the hash value produced by the given input parameter values and the hash value of the contents of the files referenced if such reference parameters exist.

3. Workflows for Intelligent Monitoring

Using the caching functionality as implemented by the proxy services, it is possible to implement a monitoring tool and convert standalone workflows into data monitoring services. The basic idea is that as now the repeated execution of a workflow on identical or similar data has very low overhead, we can easily repeatedly execute the workflow and check for user-specified conditions. In detail, the setup and execution of a monitoring task goes as follows:

1. The user sets up a workflow and registers it for monitoring. He identifies a Boolean output of the workflow as the notification output and one Boolean output as the termination output
2. A caching context is set up for the monitoring workflow
3. The workflow is executed by the enactor in the caching context. Note that caching does not apply for non-deterministic services. As data access operators are non-deterministic, this triggers a query for new or updated data.
4. If the value notification output is true, the user is alerted of new results

5. If the value of the termination output is true, the monitoring task is stopped, else the algorithm is repeated from step 3, possibly after a pre-determined waiting period.

The requirement of Boolean notification and termination output in Step 1 allows the implementation of more complex criteria with the tools provided by the workflow language. For the sake of usability, however, it is also straightforward to implement more complex stopping criteria for other types of output directly in the monitoring tool. Note that in Step 3 there is some optimization potential if the data access operators support a notification mechanism when new data is available, or allows querying the last update time of the underlying database.

A possible drawback of this implementation is the excessive use of caching. However, an additional advantage of the proxy services is that they easily allow detailed logging of service statistics. In particular, the total execution time of the proxy services, the execution time of the underlying service, the caching overhead, and the average number of calls in a given time period can be recorded. With this information, it is easily possible (1) to check if caching is effective (*caching overhead* < *execution time of underlying service*), and (2) to estimate the total reduction of execution time for each service (*(execution time of underlying service – caching overhead) * number of calls*). Should the size of the cache exceed a pre-determined threshold, caching can be de-activated for services with the least total reduction of execution time.

4. Experiments

In order to illustrate the gain in efficiency in using caching, a scenario based on simulated clinical trial has been used. In this scenario, two research centers are enrolling patients in a clinical trial on breast cancer (hereafter called the Multi-Center Multi-Platform or MCMP scenario). Both centers use different microarray platforms to assess genome-wide gene expression in biopsies taken from the patients. A single microarray is used for each biopsy, which yields up to millions of individual gene-expression measurements. In order to compare gene-expression measurements for biopsies obtained on a given microarray platform, a data-normalization procedure involving many (usually all) measurements that were made with this platform is required. Thus each time a new patient is enrolled in the study, the computationally costly normalization procedure has to be repeated.

Now, in the scenario two platforms are used. The gene expression measured for a biopsy collected in one of the centers will not be involved in the normalization procedure for the data collected in the other center, which can thus be avoided.

Figure 3 shows a possible implementation of a workflow which would be typical of such a monitoring procedure. Data are flowing from top to bottom, with files as inputs and each horizontal block in the figure representing a GridR service [7]. The workflow has vertical symmetry, with the left half corresponding to the data preprocessing for one microarray platform (one research center) and the right one for the other. The left-most and right-most input blocks at the top represent the microarray data. (The upper-center blocks are related to microarray annotation and are not relevant in the present scope.)

Microarray data collected in both centers are combined with the clinical data collected from the patient (age, gender, etc.) in the trial monitoring block of the workflow (bottommost block in Figure 3). Thus when a new patient is recruited in the

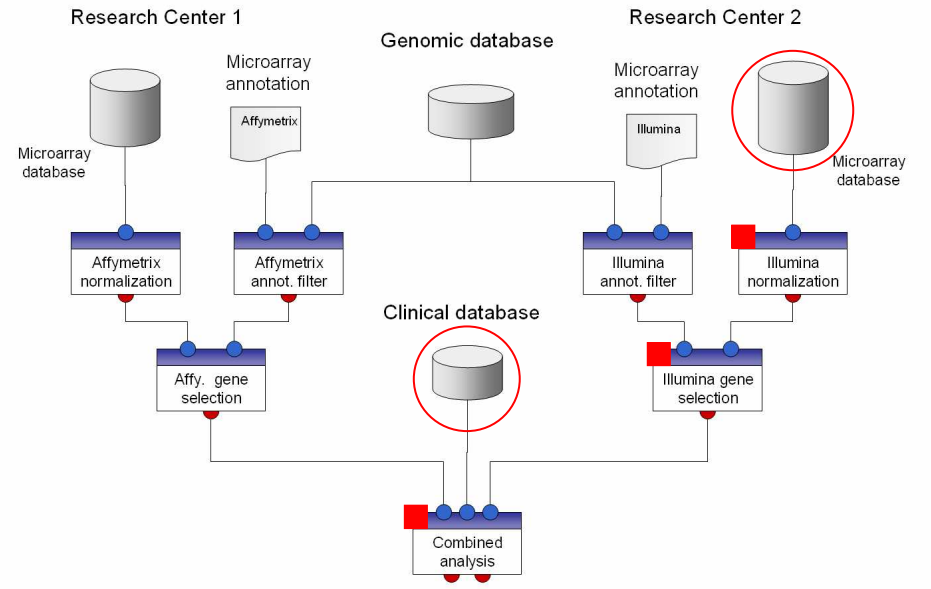

Figure 3. Possible workflow for continuous monitoring of the clinical data collected in the MCMP scenario. The red circles indicate the data inputs that are affected by the recruitment of a new patient in one of the research centers. The workflow components marked with a red square are those for which a new execution is required. See text for more details.

trial, two inputs of the workflow are affected: one in either of the microarray-data related branch, and one corresponding to the clinical data. An example of patient recruitment is represented by red circles on the figure. The workflow component requiring an update in this case are marked by red squares, the intermediate output of all other components being unaffected by the new data.

For the practical application of the concept described here, two microarray datasets representing 73 patients each have been used, with the corresponding clinical data. The microarray platform were Affymetrix HG-U133A with about 500'000 features, and Illumina, with about 23'000 genes measured (summarizing about 600'000 array features). The microarray-annotation related blocks identify which array features are truly representing known genes, based on the latest gene-sequence annotation. The feature-filtering blocks identify which array features are the most informative for a given genes. Finally the trial monitoring block assesses the significance of a gene-expression-based sample classifier.

In order to show up the possible performance improvements with caching, we evaluated the runtimes of the services in the workflow shown in Figure 6 in the ACGT system. For practical reasons, the underlying databases had to be replaced by file inputs. However, as these data sources are non-deterministic, their execution time is identical in both a caching and non-caching scenario anyway.

With respect to caching, the important properties of the workflow are that the left part (Affymetrix) and right part (Illumina) can run in parallel. Each of the parts consists of two steps, where the gene selection can only start if normalization and annotation filtering is finished. The bottom part (combined analysis) can start when both gene

selection steps are done. In addition, all services of the workflow except the data access services are deterministic.

With respect to the ACGT system it is important to notice that the services receive large inputs and outputs as references to files stored in the Grid Data Management System (DMS). The DMS guarantees that the contents of a file cannot be changed after being written, such that identical identifiers guarantee identical content (but not the other way round). As discussed in Section 2, other implementations may require different ways of efficiently checking the identity of inputs. However this does not affect the validity of the overall caching approach.

In consequence, checking the identity of the content of files in the ACGT system works in two phases: If the identifier of the corresponding file was cached, the input is identical (reference check). If not, the file must be downloaded and its content must be compared to the cached file (download check). If the content of all inputs is identical to the values in the cache, we have a cache hit and can directly return the cached output value without calling the actual service. If not, the service needs to be called and the cache is updated.

For the comparison of the references we have a constant time of 1s for the check. For the downloading and hash value comparison we assume a time of 2s plus the time it takes to download the file. The caching is computed on a single machine where the proxies are running. We assume that the time for caching is serial time where the caching checks cannot be computed in parallel.

The Affymetrix and the Illumina Microarray annotation files are candidates for taking a reference directly from the cache, as these files always have the same identifier. The outputs of the queries to the databases are always stored in separate files with different identifiers and thus have to be compared each time.

Caching can save time in case an input can be found in the cache and the computation time for the check is less then the time it takes to run the service. But, of course the caching causes overhead in case we have a cache miss so that the check of the cache as well as the service have to be computed. In the following, we distinguish between 3 different scenarios:

1. **Workflow execution without caching:** The overall workflow had a total runtime (wall clock time) of 39m19s while the services run partially in parallel. The serial runtime (CPU time) is 58m14s.
2. **Workflow execution with caching on the same data:** Assuming that none of the data sources changed, the execution of the workflow using caching is as follows. Depending on the specific workflow component and their inputs, caching by reference check or caching by download and direct comparison is used. Once all inputs of a service come from the cache, the service is not executed and a reference cache file is directly used as output. As the data does not change, caching can be used for each service. The CPU time and the Wall time, as the caching is computed sequentially on single machine, is 2m43s.
3. **Workflow execution with caching on partially new data:** In this case we assume that a new patient's data arrives in the Illumina database. In our example, the data sources "clinical database" and "microarray database" change due to a new patient coming in. Thus, three of the services have to be computed, for the others caching can be used. The Wall time was 34m54s and the CPU time 36m58s.

Table 1: Execution times of the workflow in the three caching scenarios. As services can run in parallel, we distinguish between CPU time (time spent on the calculation) and Wall time (for the overall runtime). All times given in the format mm:ss

	No caching		Same input		Part diff input	
Name	CPU time	Wall time	CPU time	Wall time	CPU time	Wall time
Illumina normalization	03:59	03:59	00:23	00:23	04:23	04:23
Affymetrix annot. filter	03:31	03:59	00:17	00:40	00:17	04:23
Affymetrix normalization	08:21	08:21	01:26	02:06	01:28	04:23
Illumina annot. filter	03:31	08:21	00:17	02:23	00:17	04:23
Illumina gene selection	07:54	16:15	00:02	02:25	08:20	12:43
Affy. Gene selection	09:25	17:46	00:02	02:27	00:02	12:43
Combined analysis	21:33	39:19	00:16	02:43	22:11	34:54
Sum:	58:14	39:19	02:43	02:43	36:58	34:54

Table 1 gives an overview over our experiments on the workflow execution. As example for the time measurements, let us have a look on Step “Illumina gene selection”. In case we do not use caching, the service has to be executed which takes a time of 7m54s. If we execute the step with caching, but the inputs did not change, we have a time of 1s each for the reference check of the left and right input. If the input changed as explained, we still have a time of 1s for the reference check of the left input, a time of 25s for the reference and download check (download 22s, service call and check 2s) of the right input and a time of 7m54s for the service execution.

We also compare the runtimes of each individual service in 4 scenarios (see Table 2). First using no caching at all, second the best case for caching (all inputs are references), third the worst case while still having cache hits (all inputs have to be downloaded) and fourth the worst case using caching which is having cache misses for each input. Note that the computation time of the service depends on the specific input.

Table 2. Results of individual service executions. All times given in the format mm:ss

	No caching	Cache hits (reference) for all inputs	Cache hits (download) for all inputs	Cache miss for all inputs
Name	time	time	time	time
Illumina normalization	03:59	00:01	00:24	04:23
Affymetrix annot. filter	03:31	00:02	00:53	04:24
Affymetrix normalization	08:21	00:01	01:28	09:49
Illumina annot. filter	03:31	00:02	00:34	04:05
Illumina gene selection	07:54	00:02	00:41	08:32
Affy. Gene selection	09:25	00:02	00:39	10:04
Combined analysis	21:33	00:03	00:54	22:27

By using caching users can profit from better total performance or usage of less resources for the computation of a workflow. As caching causes overhead, the overall profit depends on the likelihood with which the input data changes (and the frequency of the workflow executions). It is obvious that for short running services caching performs worse, but for long running services it is useful. Our experiments show that the parallelization of the service execution saves computing time. In our example, caching improves the performance even more. In case the data sources do not change,

the service execution is a lot faster then the normal execution. In case some of the services have to be executed, the overall computation time was still a bit lower than the normal execution, and the time spend on the CPUs was much lower. Thus, depending on the changing on the input data, caching can improve the efficiency of workflow execution with respect to both computation time and usage of computing resources. However, the caching works only for services generating deterministic outputs.

5. Conclusions and Future Work

In this paper we presented the investigation on how to seamlessly implement the step between a one-time proof-of-concept workflow and high-performance on-line monitoring of data streams in the context of data monitoring. The use case of our study was a long-running clinical trial. We introduced an approach based on proxy services that allows executing single-run workflows repeatedly with little overhead.

A special case that is not considered in this paper is for services that are non-deterministic because they are random or are subject to eventual numerical errors. For these, it would be possible to introduce more complex tests in the proxy services, such that a cache hit is already produced when the results are identical up to some tolerance of error. However, these tests would be very much dependent on the data types and the numerical requirements of the workflow. Another interesting point of research would be to investigate a possible shared caching of services of multiple workflows.

Acknowledgements
Christine Desmedt and collaborators at Institut Jules Bordet, Bruxelles, and Francesca Buffa and collaborators at University of Oxford have provided the clinical and research data used in the MCMP scenario. Francesca Buffa provided useful advice in the development of some of the workflow components. In addition, the authors gratefully acknowledge the financial support of the European Commission for the Project ACGT, FP6/2004/IST-026996.

References

[1] I. Foster. *The Grid: Computing without bounds*. Scientific American, 288(4):60-67, 2003.
[2] B. Claerhout, N. Forgó, T. Krügel, M. Arning, and G. De Moor, *A Data Protection Framework for Transeuropean genetic research projects*, Studies in health technology and informatics, 2008.
[3] G.C. Fox and D. Gannon. Special Issue: Workflow in Grid Systems: Editorials. Concurrency and Computation: Practice & Experience, 18(10):1009-1019, 2006.
[4] M. Tsiknakis, M. Brochhausen, J. Nabrzyski, J. Pucacki, S. Sfakianakis, G. Potamias, C. Desmedt, and D. Kafetzopoulos. *A Semantic Grid Infrastructure Enabling Integrated Access and Analysis of Multilevel Biomedical Data in Support of Postgenomic Clinical Trials on Cancer*. Information Technology in Biomedicine, IEEE Transactions on, 12(2):205-217, 2008.
[5] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke, *Security for Grid services*, Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, 2003., 48-57.
[6] P. Louridas, *Orchestrating Web Services with BPEL*, IEEE Software, vol. 25, no. 2, pp. 85-87, March/April, 2008
[7] D. Wegener, T. Sengstag, S. Sfakianakis, S. Ruping, and A. Assi, *GridR: An R-Based Grid-Enabled Tool for Data Analysis in ACGT Clinico-Genomics Trial*,. Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing (eScience 2007), Bangalore, India, 2007, pp. 228-235.