

A question of scale

Bringing an existing bio-science workflow engine to the grid

Stefan Frank, Josh Moore, and Roland Eils

intelligent Bioinformatics Systems
German Cancer Research Center
Heidelberg, Germany

1 Introduction

The bioinformatics community is characterized by an extreme interdisciplinarity—biologists are working together with mathematicians, who are working together with physicians, who are working together with computer scientists. Along with this heterogeneous background comes a plethora of different approaches to modeling bioinformatic workflows. The tools used to tackle this challenge have long been dominated by proprietary products covering very specific areas of use. What's even worse, a variety of proprietary formats for describing these workflows have increased the tendency to form isolated environments.

Recent successes in Systems Biology and increasingly sophisticated lab-practices like RNAi[1] confront the biologists with various problems of scale:

- the amount of data that is produced and needs to be analyzed is dramatically increasing
- the complexity of workflows increases as new biological mechanisms are uncovered
- the need for national and international cooperations increases to unite and coordinate resources on hard-to-solve problems

For most areas of bioinformatics, open standards for the exchange of data are currently emerging[2],[3], but attempts to standardize bioinformatic workflows [4] have not yet found widespread adoption. One of the possible reasons seems to be the heterogeneity of the community and the wide range of problems covered by bioinformatics.

Although current grid infrastructures like Globus[5] show promise in addressing many of the problems scalable virtual organizations are confronted with, the area of grid-enabled workflows still seems to be characterized by a lack of implementation or conflicting approaches. As there is yet no clear winner to come out of various efforts like [6], and [7], we decided upon a very pragmatic approach to grid-enable our own proprietary workflow engine.

Web services currently form the most stable and most agreed upon standards in this field. We decided, therefore, to move our tools consequently into using established protocols like WSDL[8], SOAP[9] and BPEL[10], that are not only backed by the heavy-weights of our industry like IBM and Microsoft, but also by independent tool-vendors and the open-source movement. We embed these projects into our long-term goal to adapt the OGSA – efforts on standardizing grid-infrastructure, as soon as the standards and tools mature.

The tool used at *iBioS* – *mine-it* – to visually model and process bioinformatic workflows has been a joint effort between *iBioS* and *phase-it* (now Europroteome, [www.europroteom.org]). Although very successful in its early stages, we were soon confronted with many issues of scale. Here, we present different strategies to overcome these issues and a pragmatic way to migrate existing models and tools to a standardized, grid-enabled environment. We evaluate three methods:

- porting the application to a J2EE-Architecture and using JMS to connect to compute-agents,
- encapsulating workflows, distributing them using standard job-schedulers and making these workflows accessible as web services, and
- extending these web services-approach to embrace emerging workflow standards for web services[10].

We will show pros and cons of these approaches and from there derive some general guidelines on how to adapt legacy-workflow systems to a grid-environment.

2 Migrating a legacy workflow system

Historically, *mine-it* was devised as a stand-alone tool for modelling bioinformatic workflows especially for data-analysis and classification. Much effort has been put into making advanced analysis-tasks (neural networks, genetic algorithms, decision trees) easily accessible for the biologists. Based on these tasks, complex workflows for the analysis of gene-expression and microarray-data have been built by our group and collaborators.

As a stand-alone tool, *mine-it* has some distinct advantages and just as distinct drawbacks.

2.1 Pros

Ease of use A simple workflow model makes *mine-it* a tool well-suited for the biological scientist without higher degrees in information science. Nodes representing basic bioinformatic building blocks are connected by arcs transporting a common data format.

Interactivity Developing bioinformatic workflows involves many incremental cycles of exploration, testing, training, and verification. In the locally running *mine-it* instance, if the data volume is kept low and demands on computing are reasonable, the lack of a clear distinction between design-time and run-time of a workflow provides a seamless and highly interactive experience for the scientist.

2.2 Cons

Most issues with *mine-it* involve problems of scale.

Memory and processing power Since memory and processing power is limited by the client's machine, *mine-it* cannot cope with the dramatic increase in available microarray data. It also proved not suitable to run advanced, resource intense computations, like evolving neural network classifiers with genetic algorithms or the C5.0 decision trees method. Further, the lack of a clear distinction between workflow design-time and run-time proved a hindrance in easily distributing tasks.

Interoperability with non-Java clients Since *mine-it* is a pure Java application, it is very easy to extend its functionality using Java. However, the programming landscape in biology tends to be extremely heterogeneous. Apart from popular scripting languages like Perl and Python, statistical and mathematical packages like R or Matlab play an important role. Although it is possible to integrate Java into all these computing environments, actually doing this is more work than our group can reasonably handle.

Sharing workflows in Large-Scale Collaborations The effort put into the design of a workflow for the analysis and classification of microarray data is immense. These workflows capture the essential biological and data-mining knowledge of our group – unfortunately, with the original version of *mine-it* sharing this knowledge with our collaborators was not easily possible. It required the use of the *mine-it* framework, and therefore the sharing of the *mine-it* code.

Exchanging software artefacts (e.g. program binaries, database schemas, schemas, etc.) is cumbersome and hard to manage. Issues like software updates, schema migration, and help-desk services simply cannot be handled acceptably with the resources of a scientific organization.

2.3 Requirements

This situation led us to a project with the following requirements:

- Make use of a commodity cluster (20 dual-processor HP NetServers running Debian/Linux) to run workflows in-house. The cluster shall be used to parallelize and distribute workflows to solve the problems of lacking memory and processing power.
- Make workflows sharable and enable cooperation at a service-level.
- Preserve the efforts that went into *mine-it*. This includes the specific tasks that the *mine-it* platform can run (neural networks, SVM, etc) but also the tested and proven workflows.

In order to achieve this, we tried three different approaches:

- Devise a complete J2EE-based architecture that accepts and stores workflows and offers an engine that distributes the workload upon compute agents running in the cluster.

- Encapsulate finished workflows and add a Web service interface to them. This approach introduced a clear distinction between run-time and design-time. Workflows were designed and tested using the stand-alone tool, then encapsulated, and afterwards scheduled and distributed by a third-party scheduling engine. Afterwards, Web services were added to the encapsulated workflows, allowing for parameterization, submission, cancellation and monitoring of workflows.
- Encapsulate nodes into services and use BPEL to bind them together. We recently started this subproject to add web services of more fine grained units. Instead of encapsulating whole workflows, we use small parts of workflows or even nodes as a base for our services. We are currently evaluating different tools to tie these small units together using BPEL.

3 Methodology

3.1 J2EE Solution

With our first approach we used a standard J2EE-Architecture to migrate *mine-it* from the stand-alone to a client-server version. Workflows were built in the client but run on the application server. The work was distributed to several compute-agents using JMS. Along with the distribution of parameters and data, we exploited standard Java mechanisms to distribute the code needed to run the computations on the compute-agents. This helped to resolve some of our scaling issues:

- Sharing of workflows: With all workflows stored in a central repository, they can easily be shared.
- Distribution of workload: The compute-agents that do the actual processing are distributed on the cluster. As they are identical, new agents can be added or removed without disrupting the system. By clustering the central application server which drives the workflow, one can also eliminate this single point of failure for a redundant, failsafe and highly distributed system.
- Sharing of infrastructure: Since the actual computations run on the compute-agents and not on the client machine, the amount of code for the client was significantly reduced. The client became mainly a viewer and a workflow designer – these are easily distributable using Java Web Start(JWS). Maintaining the infrastructure for JWS significantly reduced our workload for the software-distribution.
- Interactivity: For the client, not much has changed. Activating a node instantly dispatches it to an idle compute-agent. The node is processed immediately and results delivered to the client. In cases where the machines running the compute-agents were significantly faster than clients, this resulted in an even more responsive environment for computationally intense tasks.

However, this approach had some drawbacks:

- **Data routing:** Having an application server as the central driver for the workflow implies routing data physically from one machine to another. In the current version, all data needed for computations flowed through the central application-servers. As the amount of data increases, this becomes an obvious bottleneck.
- **Difficult transactional scenarios:** Due to the tight coupling of the nodes inside a workflow, the transactional behavior of such a system rapidly becomes very complex. The tight transactional boundaries usually used in J2EE systems for complex workflows quickly resulted in high load on the transaction manager and numerous deadlocks. For a detailed discussion about tightly-coupled versus loosely-coupled transactional scenarios see [14].
- **Quality of Service:** Jobs are dispatched with a simple first-come, first-served scheme. If a compute-agent is idle, he registers interest in a job-queue, and a job will be dispatched to him for execution. Machines are not currently qualified in terms of service data (number of processors, physical memory, disk space, etc.) - querying this information from a pure Java environment proves to be difficult but possible.[15] The setup of JMS-queues that respect this service-data quickly becomes complex. Clearly, JMS is not tailored to handle this QoS-data.
- **Scaling to global cooperations:** As the tool originated from a stand-alone version, security has not been much of an issue. The demands on security in biological/medical workflows, however, can be extremely high. For example, this becomes a central issue when patient data is traveling over the wire. In the current setup, this is solved solely by limiting the collaborations and clearly separating the data the workflows work on.
- **Scaling to other clients:** The tight coupling of the design-client and the application server did not really help in making it easy to integrate the systems with other clients.

3.2 Web Services and Code Generation

These difficulties over the past several years and the sheer complexity of the system that was emerging led us to the examination of current technological trends, most notably SOAP and WSDL. For the current authors, the foremost goal has been the immediate access to our workflow engine by outside users.

To accomplish this, we developed a multi-layered scheme, shielding the workings of our internal applications from external users while also standardizing the interfaces. Within *iBioS* there is a wide-range of programming environments: the computational cell biologists prefer C/C++; the modelers and simulators Matlab; the data miners R. Providing simple access to their various applications and at the same time making them run on a high-performance cluster was a task currently without fixed-form solution. This goes doubly for an application like *mine-it* which itself is a programming environment.

Therefore we broke down the entire complex task into smaller sections where solutions currently exist: access, execution/scheduling, and finally parallelization.

3.2.1 Access

As mentioned, we examined SOAP as a next possible solution for the access layer. This integrated well with our current infrastructure: Apache's Tomcat was already running on many of our servers, and the availability of the AXIS toolkit made the step quite manageable. The access layer, therefore, consists of a web service which handles authentication, persistence, and the submitting of a task to an underlying Cluster Engine object.

All applications receive a similar interface which is characterized by a XML schema defining the necessary inputs and the expected outputs. In the case of the *mine-it* application, this schema details the necessary data inputs—usually no more than a few data files, plus the data outputs. This web service has been made available optionally through a web portal [13] or any number of clients—[16], [17], [18], for example.

3.2.2 Execution and Scheduling

Once the Cluster Engine object receives a task, it invokes the Java wrappers related to that specific task type in order to create a job submission script. This is then submitted to the cluster which calls the native applications. The user receives an email upon completion of the task or can check with the web service to see if the task is completed. An XML representation of the output can be received via either HTTP or SOAP.

3.2.3 Parallelization

Parallelization, however, remains an area where currently much work must still be done manually. In the bio-sciences, it is not enough to simply process a workflow once, but often it will need to be repeated many times to reduce bias (e.g. with n-fold cross validation) and/or to reduce variation (e.g. averaging, ensemble or voting schemes).

To accomplish this on the Sun Grid Engine, the true native application—the workflow engine—is further wrapped in cross-validating scripts which are directly called by the cluster. Although this approach works, the long-term goal will be to enhance workflows with enough meta-data to be able to derive a parallelization that fits both the problem domain and the available resources. We are currently examining approaches for automated distributions[19], especially to extend these methods on models using unstructured grids[20].

3.3 BPEL

Encapsulating complete workflows behind web services proved to be a very valuable step towards making the expertise of our group easily accessible to collaborators. However, such coarse-grained building blocks are not easily reusable. The next step we are focusing on is manually determining finer-grained units that can be more easily combined. We are currently using BPEL to orchestrate these units. This allows the end-user to use off-the-shelf editors, for example Collaxa BPEL Designer or BEA's Weblogic Workshop, to define distributed workflows containing our sub-workflows.

4 Results

4.1 Complexity of J2EE

Although we achieved some very fine results concerning the distribution of workflows to the cluster, the drawbacks in the end were overwhelming. We managed to extend the scope of the solution to other organizations inside the DKFZ, but due to a lack of working and extensible infrastructure, we see no hope to extend this solution beyond our organizational boundaries - at least not with justifiable effort.

Although the blurring of the boundary between design-time and run-time proved to be valuable for the development of complex workflows, it showed to be a major hindrance in tackling problems of large data volume or complex modelling and simulation tasks.

The tight transactional coupling that was introduced showed to be fatal for complex, distributed workflows. Although this transactional security is needed to ensure the consistency of the results, the concept of a workflow as one large transaction does not lead to scalable results.

The lack of efficient tools to manage data also proved to be a problem. Shifting around large quantities of data using J2EE mechanisms alone is not an option - the alternative is a mixture of mechanisms to manage different kinds of data-stores.

The lack of sensors (load, memory) and some properties of the Java Virtual Machine (e.g. delays in memory usage, "start-up") produced difficulties in dispatching jobs with standard schedulers, resulting in overloaded or idle machines in the cluster.

Finally, the lack of integrability with non-Java clients severely limits the scope in which our workflows can be used. Although Java has found widespread use in bioinformatics, the need to integrate with other languages will not go away.

To sum these efforts up: the approach showed quick, acceptable results, but the efforts to expand this to a fully scalable and manageable solution is something that far exceeds the reach of our organization.

4.2 Sufficiency of Web Services and Code Generation

Initially, we were only able to provide finished workflows to the community. There was an existing demand for a small number of well-tested workflows within our community, so this was a logical first goal. With minimal effort, these were encapsulated as a single process needing 4-10 parameters and a data file as input. These workflows provided a good introduction to the capabilities of the system and allowed one researcher to quickly compare multiple classification methods. Each process ran on a single node of our internal cluster.

Next, we were able within the same framework to accept any valid workflow created with the original client. Users could install the system locally, save workflows, and then submit them for processing. Workflows are saved with an XMLEncoder so that a workflow is nothing more than a representation of the graphical object. A workflow which looks for all of its input data within the directory of execution can then be submitted along with an attachment of necessary files for full execution.

The many layers of this solution are not optimal but certainly sufficient with respect to current technology. What we would like to see would be a single system with which can tackle each of our subproblems without manual interaction. We have found that such a solution is still lacking.

The ad-hoc approach to workflow distribution presented here leaves us with several infrastructure issues—security, data distribution, distributed process scheduling. Many of these issues fall away with the existence of a large-scale Globus infrastructure, which is being built with the hope of interacting with the future D-Grid. With this infrastructure in place, the role of BPEL will certainly increase.

5 Conclusion

What we have described here is one group’s journey towards the grid-enabling of a legacy workflow system.

We evaluated, built, tested, and eventually rejected a J2EE architecture, not because of any inherent failings but simply because the nature of the project was not conducive to our group. If a group is thinking of going this direction, they should do it committedly—it requires significant resources.

For us, a multi-layered conception of workflows has proven simple and rewarding. Initially, a coarse-grained model is all that is feasible. With new technologies and new methods, an ever-finer grained distribution will be possible. This is made possible through the standards behind web services, BPEL, and Globus. Implementing these technologies is beneficial and, most importantly, can be done incrementally. Adhering to such standards allows scientific organizations to significantly reduce development efforts, focus on their research tasks, yet still cooperate on a global scale.

6 Acknowledgements

The development of *mine-it* has been an extended effort in the *iBioS* group. The authors would like to thank the original *mine-it* developers, Johannes Fieres, Dr. Jan Weimer and Alex Proudkin, now members of Europroteome. Similarly, the efforts of the distributed *mine-it* team, Dr. Wolfgang Tvarusko, Armin Groll, and Ming Xu, helped significantly to illuminate the path that needed to be taken. Finally, the current members of *iBioS* have proven especially generous in providing applications for the grid efforts. Many thanks.

References¹

- [1] Bartel, David P. “Review MicroRNAs: Genomics, Biogenesis, Mechanism, and Function,” *Cell*. 2004; Vol. 116(2): 281-297.
- [2] MAGE Group, The. “MicroArray Gene Expression Markup Language (MAGE-ML),” <http://www.mged.org/Workgroups/MAGE/mage.html>

¹All URLs were checked for availability on January 23, 2004.

- [3] MAGE Group, The. “Minimum information about a microarray experiment (MIAME),” <http://www.mged.org/Workgroups/MIAME/miame.html>
- [4] Finney, et al. “Systems Biology Workbench (SBW),” <http://sbw.sourceforge.net/>
- [5] The Globus Alliance “The Globus Toolkit 3.0,” <http://www-unix.globus.org/toolkit/download.html>
- [6] Workflow Management Coalition “Workflow Process Definition Interface – XML Process Definition Language (XPDL),” http://wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf
- [7] Krishnan S., Wagstrom P. von Laszewski G. “GSFL : A Workflow Framework for Grid Services,” <http://www-unix.globus.org/toolkit/download.html>; (Draft), July 2002.
- [8] World Wide Web Consortium “XML Protocol Working Group” <http://www.w3c.org/2000/xp/Group/>
- [9] World Wide Web Consortium “Web Services Description Language (WSDL) 1.1” <http://www.w3.org/TR/wsdl>
- [10] BEA, IBM, Microsoft, et al. “Specification: Business Process Execution Language (BPEL) for Web Services Version 1.1 05 May 2003,” <http://dev2dev.bea.com/technologies/webservices/BPEL4WS.jsp> and <http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/> and <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbiz2k2/html/BPEL1-1.asp>
- [11] Europroteome. <http://www.europroteome.com/>
- [12] Tvarusko, Wolfgang. “Towards An Integrative Toolbox for Computational Cell Biology using Grid-Resources,” HealthGrid, Lyon, 2003.
- [13] Cocoon. “SOAP LogicSheet,” <http://wiki.cocoondev.org/Wiki.jsp?page=SOAPLogicsheet>
- [14] Mikalsen, Thomas, Stefan Tai, and Isabelle Rouvellou “Transactional Attitudes: Reliable Composition of Autonomous Web Services,” IBM T.J. Watson Research Center, Hawthorne, New York, USA. <http://www.research.ibm.com/AEM/pubs/wstx-WDMS-DSN2002.pdf>
- [15] Laboratorio d’Informatica Avanzata. “Monitoring Application Programming Interface (MAPI),” <http://lia.deis.unibo.it/Research/ubiQoS/monitoring.shtml>
- [16] The Apache Software Foundation. “Axis User’s Guide,” <http://ws.apache.org/axis/java/user-guide.html>
- [17] van Engelen, Robert A. “gSOAP: C/C++ Web Services and Clients,” <http://www.cs.fsu.edu/engelen/soap.html>
- [18] Kulchenko, Pavel. “SOAP::Lite for Perl,” <http://www.soaplite.com>

- [19] Kaufmann J., Lehmann T. “OptimalGrid: The Almaden SmartGrid project. Autonomous optimization of distributed computing on the Grid,” IEEE TFCC Newsletter, Vol. 4, No. 2
- [20] Chung, T.J. “Computational Fluid Dynamics” Cambridge University Press. January, 2000.
- [21] D-Grid. “Initiative zur Foerderung eines Grid-basierten e-Science Frameworks in Deutschland,” <http://www.d-grid.de/>