SERVICE WORKFLOWS AND DISTRIBUTED COMPUTING METHODS FOR ^{13}C METABOLIC FLUX ANALYSIS

Tolga Dalman[1], Ernst Juhnke[2], Tim Dörnemann[2], Michael Weitzel[1], Katharina Nöh[1], Wolfgang Wiechert[1], Bernd Freisleben[2]

[1]Institute of Biotechnology 2, Forschungszentrum Jülich, 52425 Jülich, Germany
[2]Department of Mathematics & Computer Science, University of Marburg, 35032 Marburg, Germany

t.dalman@fz-juelich.de (Tolga Dalman)

Abstract

In the field of Metabolic Engineering and Systems Biology, a plenitude of modeling and simulation tools have recently emerged that aim to shed light on processes in living cells. ^{13}C Metabolic Flux Analysis (13C-MFA) is an evolved, model-based approach for the quantification of intracellular fluxes, the key functional output of metabolism. To increase flexibility in 13C-MFA application design, the automation of repetitive tasks and the management of computational resources based on a scientific workflow framework entails many advantages.
In this contribution, we present two important aspects regarding the integration of 13C-MFA applications into a generic Scientific Workflow System (SWS). The first aspect addresses the unification of the various data formats involved in a typical 13C-MFA application, namely experimental data, models, model parameters and simulation results. Secondly, existing simulation tools need to be extended by a web service interface to make workflow orchestration possible. Hence, by making use of service-orientation, SWS technology offers many benefits for 13C-MFA as a major tool for Metabolic Engineering.

Keywords: 13C-MFA, Scientific Workflows, BPEL, SOA, Distributed Computing

Presenting Author's Biography

Tolga Dalman received the diploma degree in computer science from Bielefeld University in 2005. His main research interests include Scientific Workflows, Grid Computing, applications in the field of Systems Biology and statistics.

1 Introduction

In recent years, Metabolic Flux Analysis with ^{13}C-labeling experiments (13C-MFA) gained more and more interest as a model-based diagnosis tool in the field of Metabolic Engineering and Systems Biology [1]. It is currently the most powerful and widely applied method that allows quantification of intracellular conversion rates, so called metabolic fluxes.

To estimate model parameters, *in-vivo* and *in-silico* data have to be compared. The general procedure of 13C-MFA is depicted in Fig. 1. In a carbon labeling experiment, living cells are fed with specifically ^{13}C-labeled substrates. Labeling patterns in diverse metabolites are detected with highly sensitive mass spectrometry devices. In order to determine the not directly observable *in-vivo* fluxes, the real experiment is modeled *in-silico* based on initially guessed fluxes. By systematic rearrangement of the assumed fluxes, the real flux distribution is estimated by fitting the simulated data to the real ones, i.e. by minimizing their difference (parameter fitting process).

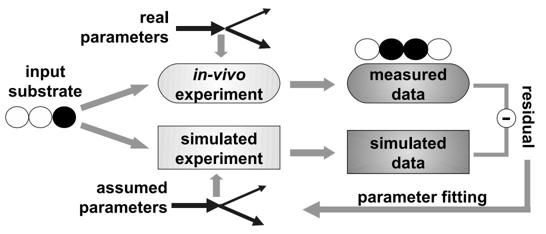


Fig. 1 The general principle of 13C-MFA [2]. Definition of interfaces between raw and simulated data, models, model parameters is required.

Although the overall procedure is conceptually simple, the mathematical core problem of 13C-MFA consists of the solution of a nonlinear inverse problem. This inverse problem is high-dimensional, and depending on the experimental data, not necessarily well-posed. As a result, determination of fluxes from labeling experiments requires a concatenation of many processing steps that are tightly interwoven and computationally demanding. Although high-performance algorithms for 13C-MFA have been developed recently [3], setting up a workflow-based 13C-MFA framework is far from being a standard procedure.

Typical 13C-MFA applications are the simulation of a carbon labeling experiment (*fwdsim*), the parameter fitting process (*fitfluxes*), or making use of criteria for statistical evaluation of flux estimations (*MCstat*). All these applications are composed of autonomous software tools that run on distributed computational resources. However, because simulation tools are often tightly integrated into each other with respect to their input and output behavior, data exchange and conversion make up a considerable amount of implementation work within large-scale 13C-MFA applications. In a modular 13C-MFA simulation framework, the generalization of data formats, and software reuse by decoupling program interfaces can drastically reduce the design complexity of such large-scale applications. In contrast to the majority of classical bioinformatics tasks, computer-based evaluation procedures in the field of 13C-MFA are highly diverse, mainly computationally intensive and only partly routine. Since 13C-MFA is a model-based approach, scientific workflows in this domain are not data-driven but control-flow based [4].

In this paper, we investigate two aspects of 13C-MFA application integration into a generic scientific workflow system in detail: (a) the unification of the various data formats involved in a typical 13C-MFA application, handling of experimental data, models, model parameters and simulation results; (b) the extension of existing simulation tools by a web service interface to make workflow orchestration possible. Implementation issues are discussed, and, finally, an example workflow modeled by the proposed approach is presented.

The paper is organized as follows. In Section 2, the principles of scientific workflows are introduced. The design of 13C-MFA web services as components of such workflows is presented in Section 3. Section 4 provides details on the implementation of our approach. In Section 5, the developed framework is used to formulate a typical example in the field of nonlinear statistical evaluation, i.e. the Monte Carlo bootstrap procedure. Monte Carlo methods are easily parallelizable on a heterogeneous cluster infrastructure. Hence, this is a paramount example for distributed computing. Section 6 concludes the paper and outlines areas for future research.

2 Scientific Workflows

The workflow paradigm allows us to flexibly create generic workflows using library methods that are encapsulated in LEGO$^{\text{TM}}$-like building blocks. These building blocks can be arranged in a comfortable way that relieves the scientist from the need to learn technical or implementation details. The single blocks are connected via communication links to enable in between data exchange.

In particular, the motivation to implement a *Scientific Workflow System* (SWS) as a generic workflow framework for the 13C-MFA domain is driven by the following objectives:

- **Workflow Automation**: many frequently recurring (sub-)activities in 13C-MFA studies can be performed in an automated manner. Scientific workflows are employed to accomplish this task adequately.
- **Data Organization**: experimental data, models, parameters, simulation results, and literature information are to be organized centrally and made accessible via a web interface.
- **Service-Oriented Architecture (SOA)**: using a SOA design, workflows are used to create 13C-MFA applications from service compositions, i.e.

they translate associated applications into the Software-as-a-Service (*SaaS*) domain. Web services, the main building blocks of a SOA, are software systems designed to support interoperable machine-to-machine interaction over a network.

- **Distributed Computing**: service compositions may extend across platforms and virtual organizations. Computational resources are assigned automatically to requested computation tasks instead of being selected manually. Interoperability, scalability and reliability are topics of current research.

In a SOA environment, web services are regarded as components of business processes. A workflow is defined by the automation, as a whole or in parts, of its business processes [5]. Thus, the aggregation of web services to large-scale applications often entails a high automation potential. Within SOA applications, loosely coupled services usually communicate through Internet protocols for so-called *programming in the large* workflow applications [6].

In the context of workflows, the attribute *scientific* emphasizes the altered needs of workflow applications in the research domain [5]. In particular, scientific applications are usually long-running computational tasks that in general cannot be handled by traditional SOA mechanisms.

Usually, web services are called synchronously. Thus, long-running applications might yield a communication timeout. To prevent such events, the communication has to be performed in an asynchronous manner. In general, this is implemented by immediately returning a unique job handle. Passing this handle to subsequent calls of the original web service allows the access to intermediary results. Because a web service can be called by multiple clients, asynchronous communication requires to distinguish different sessions of an application.

Long-running services are addressed by the *Web Services Resource Framework* (WSRF, [7]). WSRF services (or Grid Services for short) are modeled using a factory pattern, i.e., they have an explicit life-cycle management. This management is achieved by the use of web service operations that allow the instantiation and destruction of so-called *resources* (an instance of a service). Typically, a scientific application consists of several components. In the context of scientific workflows, a workflow is composed of several web services and Grid Services, respectively. Henceforth, web services and Grid Services are used synonymously in this contribution.

3 Service Workflows for 13C-MFA

An elaborate investigation of the monolithic 13C-MFA application architecture revealed about 20 independent programs. These are formulated as encapsulated black-box systems with a specified data input (for example, a metabolic network model) and an output (i.e. simulation results), and arranged in a hierarchical manner. Within the black-boxes, individual programs are arranged in serial or parallel fashion. These programs communicate with each other by various input and output formats (see Fig. 2).

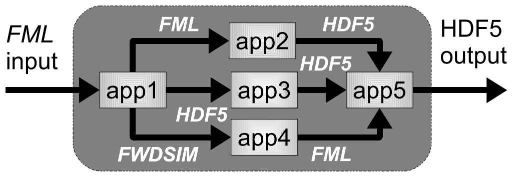


Fig. 2 Example data flow scheme: typical applications in 13C-MFA need to exchange various data files. While a user only needs to care about input (here *FML*) and output (HDF5), the whole workflow usually consists of several programs (*app1*, *app2*, . . . , *app5*) each with specific input/output semantics.

In order to make 13C-MFA applications workflow-capable, we follow two major design principles:

1. 13C-MFA programs have to be compatible to each other with respect to their input and output data formats.
2. Each 13C-MFA software tool has to be represented as a web service.

The first aspect is achieved by adapting 13C-MFA programs to restrict the use of data formats based on XML and HDF5. Secondly, 13C-MFA programs are wrapped by a web service layer utilizing the LCDL framework. A workflow orchestration engine for the automation and high-performance provisioning of 13C-MFA web services is briefly presented at the end of this section.

3.1 Interconnecting 13C-MFA Programs

Reducing the total number of employed formats significantly reduces the overall complexity of workflow applications. For example, transfer and conversion processes between the different formats are minimized. This, in turn, also eliminates potential sources of errors.

Three data formats are used in the 13C-MFA simulation toolbox:

1. Metabolic network models are represented in the XML format *FluxML*, or *FML* for short [8]. This format contains information about stoichiometry, metabolic network structure, measurement specifications and initial flux values.
2. Results from simulation and parameter estimation are stored in another XML format called *FWDSIM*. This document format includes estimated flux parameters and application run-time information, but also configuration parameters of solvers and optimizers used.
3. Bulk matrices of floating point data, such as experimental and simulated measurement data, system states or sensitivity information, are stored in

a HDF5-based file format [9]. These files are utilized, for example, whenever values from multiple *FWDSIM* documents need to be aggregated in a single file for further analysis in third-party tools such as MATLAB^{TM}.

3.2 Web Service Interfaces for 13C-MFA Tools

The Java framework LCDL (*Legacy Code Description Language*) provides an universal program wrapper [10]. Here, command line arguments of each single 13C-MFA tool are represented as arguments for a web service call. Being console- and file-oriented, extending 13C-MFA applications by web service functionality is easily achieved using the LCDL framework:

- Standard output streams (*stdout*) and standard error streams (*stderr*) are redirected into distinct files. All data is accessible by further web service requests to *stdout*, *stderr* or any other file generated from the web service. This is especially favorable for debugging and analysis purposes.
- An XML-based legacy code description language allows the definition of custom types for wrapped web services. In the 13C-MFA environment, this is a valuable feature, because workflow applications are constrained by their input/output semantics. For example, although many 13C-MFA programs have *FML* or *FWDSIM* file arguments, both XML formats have entirely different meanings, and, must not be mixed up. Thus, additional constraints prevent semantic errors in a SOA framework.
- A plug-in mechanism allows extensions to the LCDL framework, e.g, to support new message transport types such as Flex-SwA [11]. This middleware is responsible for the opaque transfer of data between subsequent web service calls, i.e. files are exchanged between 13C-MFA programs.

For the SOA integration of 13C-MFA tools, neither the learning nor the use of a different framework or language is necessary. The LCDL framework is realized as an Eclipse Rich Client Platform and utilizes several mechanisms provided by this platform, like plug-in mechanisms (using OSGi techniques) and basic editor for the LCDL model based on the Eclipse Modeling Framework.

Using LCDL for the service wrapping of existing 13C-MFA tools, and Flex-SwA for opaque file exchange between those web services, a solution for 13C-MFA workflow orchestration is required.

3.3 Workflow Orchestration with BPEL

With the availability of 13C-MFA tools in a SOA, the web services need to be organized and provisioned in an appropriate manner. Specifically, the modeling of workflows with 13C-MFA simulation tasks, and the Grid-deployment of computationally intensive jobs must be possible.

In recent years, the *Business Process Execution Language* (BPEL) has gained interest within the scientific community [12, 13], since it is – in contrast to the majority of DAG-based (directed acyclic graph) workflow languages – a Turing-complete, general-purpose workflow modeling language. In the area of scientific computing, BPEL offers a number of advantages:

- In addition to a rich vocabulary and control mechanisms to express sequences of activities like *receive*, *invoke* and *reply*, BPEL allows parallel execution, loops, error handling and compensation mechanisms to perform roll-back actions.
- BPEL is an official OASIS standard, widely used with industrial-strength tool support [14]. These tools seamlessly integrate into service-oriented and Grid-computing architectures.
- Workflows in BPEL are exposed as web services and, thus, fit naturally into a SOA. This feature supports programming in the large and interface-oriented design of complex applications.

With these properties, the integration of scientific workflow applications into a distributed workflow and Grid environment is considerably eased, and originally tightly-integrated 13C-MFA simulation tools are decoupled using web service interfaces. *ActiveBPEL*, an open source BPEL language execution engine, is chosen for the composition and orchestration of 13C-MFA scientific workflow applications [15].

4 Implementation Issues

4.1 Data Formats in 13C-MFA Applications

Applications in the 13C-MFA toolbox are written in various programming languages, such as C++, Java, Python or MATLAB^{TM}. The implementation of the data formats and exchange interfaces presented in the previous section mainly base on the following subjects:

1. The Xerces-C library is used to parse and stream XML files for *FluxML* and *FWDSIM* formats in C++. In Java, Python and MATLAB^{TM}the language intrinsic libraries are utilized for XML processing. *XML Schema* definitions for both data formats exist, thus providing validity checks on each 13C-MFA program execution.
2. HDF5 is implemented in C++ and Java using the reference library [9]. In Python, the *h5py* library is used for HDF5 processing [16], while MATLAB^{TM}ships with built-in HDF5 support. An internal interface allows the conversion to and from HDF5 of any matrix or vector data used within the 13C-MFA programs.

As a viable alternative to HDF5, the netCDF format was also considered [17]. However, as netCDF in the current version 4 internally uses HDF5, the latter was chosen due to the generality and widespread support in the scientific community.

4.2 Web Service Extension

Extending the diversity of 13C-MFA programs with a web service layer one by one is an error-prone procedure. The LCDL framework is used for the automatic generation of Java wrappers from native code, i.e. from binary executables or libraries. Each LCDL-generated web service is represented by a specification containing various information about input arguments:

- *name*: string identifier of the input argument.
- *type*: XML Schema data type, e.g. *string* or *int*.
- *mode*: flag indicating a parameter that is read-only (`in`), read-write (`out`), or both (`in-out`).

Additional specifications contain information whether an argument is optional, of type array etc. In the LCDL specification, return values of the wrapped code consist of two parts:

1. **output target**: the destination of the web service output needs to be defined, i.e. *stdout*, *stderr*, a file or an integer value are the available options.
2. **output semantics**: the specification, how the web service output is returned. For example, the *fitfluxes* program either returns an XML file containing flux estimations, or, depending on the parameterization, an integer value is returned. The semantic definition of output parameters in the LCDL code specification resolves such ambiguities.

As a generic framework for wrapping legacy code, LCDL supports different types of emitted code, so-called *bindings*. Currently, Java and web service bindings exist in LCDL. The emitted wrapper code is subject to restrictions, e.g. unlike Java, web service bindings do not allow polymorphism. LCDL emits correct code for the target binding. Because LCDL is utilized for wrapping 13C-MFA programs by web services, the emitted type has always a web service binding.

4.3 Distributed Computing

When deployed in a Grid environment, the execution of a scientific workflow is automatically provisioned onto computational resources [6]. With the availability of 13C-MFA applications as web services, the maintenance effort is drastically reduced in a multi-user environment. Hardware and software resources are managed centrally for all users in the SOA.

The main ingredients for the flexible deployment of 13C-MFA applications into a distributed SOA are made available with these deliberately selected technical components:

- Scientific workflow applications are 13C-MFA programs wrapped by LCDL-generated web services. These web services are deployed in web service container, such as Apache Axis [18].
- Several extensions to the ActiveBPEL have been developed, i.e. support for long-running workflow applications via Grid service invocations, support for *Grid Security Infrastructure*, and an on-demand provisioning component are now available [15, 19, 20].

Distributed computing and resource provisioning of SOA services in a scientific workflow environment is easily feasible with the available software tools [21]. Due to the presented 13C-MFA application infrastructure design, generic solutions, like ActiveBPEL, can be utilized in the implementation of a scientific workflow framework.

5 Sample 13C-MFA Application *MCstat*: The Monte Carlo Bootstrap

The Monte Carlo bootstrap method is utilized whenever parameters of nonlinear models are estimated with relatively few measurements in order to assess the certainty of the parameter estimates by calculating the measurement error propagation [22].

In 13C-MFA, a large number of (artificial) measurement data sets mimicking real data are generated by variation of the experimental measurements according to their standard deviations. For each artificial data set, unknown fluxes are estimated by solving the nonlinear inverse problem (parameter fitting). To address well-known pitfalls of nonlinear optimization (i.e. local versus global), we selected a multi-start strategy in connection with a gradient-based optimization routine. Several thousands or ten-thousands of Monte Carlo bootstrap iterations are performed in this way, possibly in a parallelized fashion. Here, the determination of feasible initial fluxes is a problem of its own that is described elsewhere [8].

The implementation of the multi-start Monte Carlo bootstrap application *MCstat* in the 13C-MFA environment is presented in Fig. 3. This application consists of the following 13C-MFA programs:

1. **perturb**: n measurements are sampled according to the measurement specification. The temporary FluxML files P are generated from *fml*.
2. **mciv**: a set of m feasible initial flux values are randomly sampled from the FluxML file in the flux space. The results are stored in the temporary HDF5 file F.
3. **setfluxes**: set the j^{th} flux vector from F in the perturbed model file (P).
4. **fitfluxes**: this step involves the numerical solution of the nonlinear inverse problem utilizing parameter estimation of flux values with respect to the measurements.
5. **collectfitdata**: finally, the *FWDSIM* files are merged into a unified HDF5 file (R).

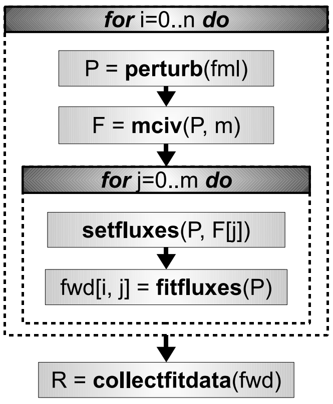
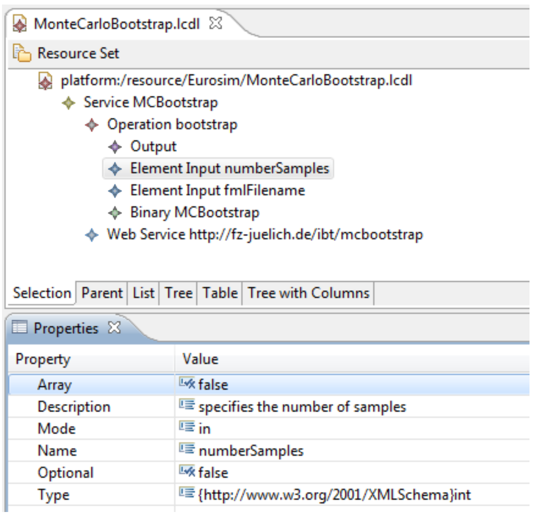


Fig. 3 Monte Carlo bootstrap application control flow *MCstat*. The input parameters to this workflow are: (1) *fml*: input FluxML model, (2) *n*: number of measurement perturbations and (3) *m*: number of random fluxes. This number of feasible initial flux values to be generated depends on the size of the flux solution space. Simulation results are then collected in a HDF5 file, named *R*.



MonteCarloBootstrap.lcdl

Resource Set

- platform:/resource/Eurosim/MonteCarloBootstrap.lcdl
  - Service MCBootstrap
    - Operation bootstrap
      - Output
      - Element Input numberSamples
      - Element Input fmlFilename
      - Binary MCBootstrap
    - Web Service http://fz-juelich.de/ibt/mcbootstrap

Selection | Parent | List | Tree | Table | Tree with Columns

Properties

Property	Value
Array	false
Description	specifies the number of samples
Mode	in
Name	numberSamples
Optional	false
Type	{http://www.w3.org/2001/XMLSchema}int

Fig. 4 LCDL model of the Monte Carlo application

As shown in Fig. 3, the Monte Carlo application consists of two nested loops iterating over the integer parameters n and m. In the outer loop, the input model fml is perturbed n times and m initial flux distributions are generated. These initial flux vectors are then iterated in the inner loop, and in total $n \times m$ executions of the *fitfluxes* application are performed. Finally, the estimated fluxes in the simulation files (fwd) are merged into the HDF5 file R. Intermediary results P, F and fwd are kept in case an error occurs. Thus, a detailed workflow error analysis is possible.

The estimated flux results in R are processed further by discarding obviously non-global optimization outcomes. In the 13C-MFA, the remaining (valid) results from the Monte Carlo bootstrap study are then analyzed in terms of statistical measures (1^{st} and 2^{nd} moments), and the a-posteriori probability distribution is determined. Finally, these results are visualized, interpreted, incorporated into new models.

Monte Carlo bootstrap methods are generally embarrassingly parallel which implies that parallelization is easy and the Monte Carlo workflow is expected to give well-scalable results. In particular, the outer loop of the *MCstat* workflow iterates over n perturbations of the input model. Within each iteration, these models are independent to each other. Except for the workflow invocation and the final data merge step of Monte Carlo results (*collectfitdata*), no communication is required. Thus, the execution of the Monte Carlo bootstrap workflow in parallel is easily possible.

In terms of service-orientation, n and m are integer arguments to the web service, the files fml and R are represented as string paths to file locations. Flex-SwA is utilized to perform the real transport of these files. The LCDL definition of the Monte Carlo bootstrap service is depicted in Fig. 4.

Exposing Monte Carlo bootstrap as a web service via invocation of a workflow script allows scientists to easily use and adapt its functionality without the need of compiling and installing any tool necessary to execute it.

6 Conclusions

We have shown that the 13C-MFA software environment can be modeled by service workflows with little effort. This step improves usability and flexibility of the modeling, simulation and evaluation pipeline and facilitates 13C-MFA for scientists without programming experience. The unification effort of data formats used by 13C-MFA programs is not only beneficial for the implementation of web services, because XML and HDF5 are standard formats available in many programming languages and tools. Using web service technology, 13C-MFA applications are easily extensible to run within a distributed computing environment, as it was demonstrated with the Monte Carlo bootstrap example.

Future work includes the integration of databases into the workflow system. Since some of the 13C-MFA programs are long-running scientific applications, these synchronous web services will be replaced by asynchronous Grid service calls. All ingredients for scientific workflow orchestration of 13C-MFA services on Grid and Cloud resources are readily available [20]. The presented Monte Carlo bootstrap workflow is to be integrated into large-scale 13C-MFA applications consisting of numerous simulation jobs and human interaction tasks.

7 References

[1] Y. J. Tang, H. G. Martin, S. Myers, S. Rodriguez, E. E. K. Baidoo, and J. D. Keasling. Advances

in analysis of microbial metabolic fluxes via ^{13}C isotopic labeling. *Mass Spectrometry Reviews*, 28(2):362–375, 2009.

[2] Wolfgang Wiechert. ^{13}C Metabolic Flux Analysis. *Metababolic Engineering*, 3(3):195–206, 2001.

[3] Michael Weitzel, Wolfgang Wiechert, and Katharina Nöh. The topology of metabolic isotope labeling networks. *BMC Bioinformatics*, 8(315), 2007.

[4] Victoria Martín-Requena, Javier Ríos, Maximiliano García, Sergio Ramírez, and Oswaldo Trelles. jORCA: easily integrating bioinformatics Web Services. *Bioinformatics*, 26(4):553–559, Feb 2010.

[5] Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, and Matthew Shields. *Workflows for e-Science: Scientific Workflows for Grids*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[6] Frank Leymann. Web Services: Distributed Applications Without Limits. In Gerhard Weikum, Harald Schöning, and Erhard Rahm, editors, *Datenbanksysteme in Büro, Technik und Wissenschaft - BTW2003*, volume 26 of *LNI*, pages 2–23. GI, 2003.

[7] OASIS. Web Services Resource Framework Specification. http://www.oasis-open.org/specs/index.php#wsrfv1.2, Apr 2006.

[8] Michael Weitzel. *High Performance Algorithms for Metabolic Flux Analysis*. PhD thesis, University of Siegen, Germany, 2009.

[9] The HDF Group. Official hdf5 website. http://www.hdfgroup.org/HDF5/.

[10] E. Juhnke, D. Seiler, T. Stadelmann, T. Dörnemann, and B. Freisleben. LCDL: An Extensible Framework for Wrapping Legacy Code. In *Proceedings of ERPAS'2009*, pages 646–650, 2009.

[11] S. Heinzl, M. Mathes, T. Friese, M. Smith, and B. Freisleben. Flex-SwA: Flexible Exchange of Binary Data Based on SOAP Messages with Attachments. In *Proc. of the IEEE International Conference on Web Services, Chicago, USA*, pages 3–10. IEEE Press, 2006.

[12] A. Akram, D. Meredith, and R. Allan. Evaluation of BPEL to Scientific Workflows. In *Proc. of the Sixth IEEE Int. Symposium on Cluster Computing and the Grid*, pages 269–274. IEEE, 2006.

[13] W. Emmerich, B. Butchart, L. Chen, B. Wassermann, and S.L. Price. Grid Service Orchestration using the Business Process Execution Language. In *Journal of Grid Computing*, volume 3, pages 283–304. Springer, 2005.

[14] OASIS. Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, 2007.

[15] T. Dörnemann, T. Friese, S. Herdt, E. Juhnke, and B. Freisleben. Grid Workflow Modelling Using Grid-Specific BPEL Extensions. In *Proceedings of German e-Science Conference 2007*, pages 1–9, 2007.

[16] H5Py Project. H5py project website. http://code.google.com/p/h5py/.

[17] The Unidata Community. Official netcdf website. http://www.unidata.ucar.edu/software/netcdf/.

[18] Apache Foundation. Apache Axis. http://ws.apache.org/axis/.

[19] Tim Dörnemann, Matthew Smith, and Bernd Freisleben. Composition and Execution of Secure Workflows in WSRF-Grids. In *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '08)*, pages 122–129. IEEE Press, 2008.

[20] T. Dörnemann, E. Juhnke, and B. Freisleben. On-Demand Resource Provisioning for BPEL Workflows Using Amazon's Elastic Compute Cloud. In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid '09)*, pages 140–147. IEEE Press, 2009.

[21] T. Stadelmann, Y. Wang, M. Smith, R. Ewerth, and B. Freisleben. Rethinking Algorithm Design and Development in Speech Processing. page (to appear), 2010.

[22] Bradley Efron and R.J. Tibshirani. *An Introduction to the Bootstrap (Monographs on Statistics and Applied Probability)*. Chapman & Hall/CRC, 1 edition, 5 1994.