A Principled Approach to Grid Middleware
– Status Report on the Minimum intrusion Grid –

Jost Berthold[1], Jonas Bardino[2], and Brian Vinter[2]

[1] Department of Computer Science, University of Copenhagen, Denmark
`berthold@diku.dk`
[2] eScience Center, University of Copenhagen, Denmark
{`bardino,vinter`}`@nbi.dk`

Abstract. This paper provides an overview of MiG, a Grid middleware for advanced job execution, data storage and group collaboration in an integrated, yet lightweight solution using standard software.
In contrast to most other Grid middlewares, MiG is developed with a particular focus on usability and minimal system requirements, applying strict principles to keep the middleware free of legacy burdens and overly complicated design. We provide an overview of MiG and describe its features in view of the Grid vision and its relation to more recent cloud computing trends.

1 Introduction

The Grid computing vision of Foster and Kesselman [9] in the late '90s promised to substantially facilitate access to remote computing and storage resources. However, today's Grid middlewares in practical use only provide a somewhat reduced model. "The Grid" falls short on expectations. Today's large-scale Grid systems demand considerable expertise from the user, maintenance is staff-intensive, and they tend to seclude their user base to a few privileged scientists. As a consequence, potential users turn away from Grid solutions today, and increasingly spend subsidies on dedicated hardware (for instance, GPGPUs). We argue that the reason for this disappointing reality is excessive middleware complexity and inapt prioritisation in its development. The Minimum intrusion Grid (MiG) started in 2004 [18] to address a number of shortcomings in existing Grid middlewares, and follows a principled approach free of inherited legacy burdens. First, MiG's principle is to minimise requirements for both users and resource providers in a Grid. Second, MiG offers more than a simple "job-shop" system: it provides a complete working environment with not just computational power but also storage, collaboration software (Wiki, forum and version control repository) and an integrated web portal that can even encapsulate whole workflows. In this paper, we give an overview of MiG, discuss its design principles and realisation, and present its advanced features for group collaboration and resource sharing. We argue for a revival of the Grid vision in view of the current *cloud computing* trend. Presenting the advanced key features of MiG, we point out that the Grid vision goes far beyond cloud ideas, and how a principled Grid middleware can be realised in a flexible and user-friendly manner.

2 Background and Motivation

2.1 Grid Vision and Grid Practice

When the term of Grid computing was coined in the late '90s by Foster and Kesselman [9], Grid was envisioned as much easier to use than conventional large-scale computing. The Grid promised transparency and single entry points to powerful parallel computing resources (computational Grids) and storage (data Grids), organised in easily managed abstract entities (Virtual Organisations). However, implemented Grid systems exposed a continuously decreasing ambition, towards a substantially reduced model for the sake of reliability and system management. And yet, using one of the existing Grid middlewares (for instance, NorduGrid ARC [6], gLite [10] or EGI [5]) which inherit code base and paradigms from the Globus Toolkit [8] still has a number of problems:

- Even simple Grid system services are very complex to administrate. As a survey of Grid middleware support mailing lists shows, middleware configuration is extremely intricate and requires considerable internal knowledge.
- A common practical problem is to reliably provide special software packages for particular users. The usual solution, manual installation and maintenance of all software on all computing resources, is error-prone, cumbersome, and totally unsatisfactory against the original Grid ideas.
- Despite it being the key feature of Grid systems, granting and obtaining access to Grid resources usually involves manual work and human factors. As a result, Grid users are more often than not privileged scientists who rely on good administrator contacts and support to suit their computing needs.
- Virtual Organisations should improve this but often create middleware incompatibilities by proprietary X.509 certificate extensions. More important, implementations of virtual organisations merely provide administrative grouping, not working environments and collaboration support.

Reasons for this unfortunate development include, among others, the inherent danger of large-scale multinational projects to produce overly complex architectures and to focus on aspects of minor interest to users. For instance, large parts of the ARC [6] middleware deal with monitoring, accounting and user roles for access control. In contrast, ARC's solution for providing software on computing resources has remained strikingly ad-hoc after years in production. The primary focus of a useful Grid middleware should be ease of use for resource users and resource providers, and maximised resource usage (including harvesting spare cycles). Besides, Grid should go beyond a mere "job shop". Users should profit from advanced middleware features tailored to better collaboration.

2.2 The Cloud Vision – a Better Grid?

Far more recent, yet substantially related to Grid, is the fashion of cloud computing – alas, a cloudy subject with a range of definitions. A restrictive one comes from Berkeley, limiting the term to flexible virtualised infrastructure provisioning (IaaS) [2]. In line with this is the widespread understanding of "cloud" as providing full user control over an entire virtual machine (as opposed to running jobs in a pre-existing setup), fostered by Amazon's "Elastic Cloud Computing" (EC2) platform (the breakthrough in cloud computing of modern imprint in late 2006). We favour a slightly broader definition by the National Institute of Standards and Technology (NIST), which also subsumes software and platform services, and emphasises that "configurable computing resources [..] can be rapidly provisioned and released with minimal management effort or service provider interaction." [14]

Be it virtual hardware or a software setup, cloud computing is characterised by minimal setup and maintenance effort, and user self-service. In essence, the goals of convenience and resource usage are the same as in Grid, and when realised merely through virtual machines, one can indeed argue that the cloud ideas are a commercialised and reduced variant of the earlier academic Grid vision. The main reduction lies in virtual organisations and group collaboration, an essential goal of Grid but completely missing from cloud computing as of today.

3 Overview of the Minimum Intrusion Grid

3.1 System Architecture

A bird's eye perspective on the MiG system (in Fig. 1) shows its central motivation: *to virtualise user access to resources*. The Grid system acts as a gateway between users and resources that provide storage or computing capabilities.

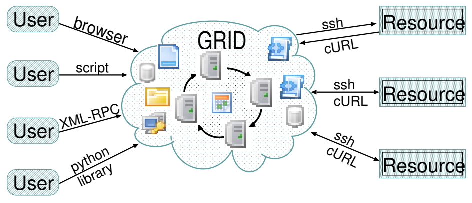


Fig. 1. Abstracted MiG Architecture

This gateway itself can be composed of replicated server instances that coordinate job execution and introduce failover safety. Users access the system via different methods, all based on secured HTTP. Communication from resources to a server uses secure HTTP as well, while a server uses SSH to address resources. The HTTP interface provides job submission and management, server file space, virtual organisations (VGrids) with shared file and web space, and means to provide and manage resources and software.

3.2 MiG Design Principles and Rationale

MiG was developed with a principled approach to follow the original Grid vision, and to address the issues observed in other Grid middlewares. This led to a number of principles and goals, ranging from technical implementation principles over architectural requirements to usability.

Non-intrusive Software Installation. The first and foremost design goal of MiG is to reduce software complexity and to avoid legacy burdens of any kind. Resources and users must be able to join and use the Grid with *minimal software* *requirements.* Experiences with other Grid middlewares show that large installations on the resource side create compatibility problems and – contrary to the Grid philosophy – require more centralised administration and maintenance. Likewise, new users may be unwilling or unable to install a large software.

Minimal Dependencies. Typical production Grid systems carry heavy burdens of legacy code and suffer from incompatibilities. Use of many different languages and tools complicates maintenance and hampers porting software to new platforms. MiG was designed to reduce maintenance requirements, and to automate maintenance operations whenever possible. This relates to the choice of implementation language as well as the tool requirements.

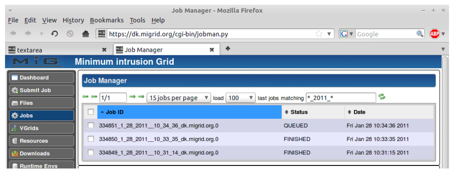
The distribution aspects of the entire system are based on secure HTTP communication. For the user, MiG only requires an X.509 user certificate from a trusted CA and a standards-compliant web browser (or other HTTP software). MiG resources require secure HTTP (cURL [4]), SSH, and a standard shell – system interfaces that can safely be assumed stable. Basing all functionality on standard software and few commonly used protocols/ports maximises MiG firewall compliance, and enables the server to update all software (scripts) on the resource without manual intervention. The MiG server software is based on the widespread Apache HTTP server and written in widely portable Python.

Fault Tolerance. Failing machines or processes within the Grid should not stop users or resources from using the Grid. If a resource crashes, the middleware should be responsible for recovering and repeating any lost jobs. The MiG design includes a concept of distributed communicating servers, where all data are replicated several times for failover safety, while keeping clear responsibilities for consistency. Jobs are replicated and quickly rescheduled by servers upon failure.

Anonymity of users and resources. Job submission in most Grid systems involves direct communication between the submitting client and potential execution resources. In such a design, a resource provider can theoretically monitor Grid usage of all users by their incoming execution requests. In the MiG design, the mediating servers can completely shield users and resources from one another if desired (but with server audit logs to trace any abuse). Assuming sufficient job throughput, it is not possible for a resource owner to identify the user who submitted a particular job. Likewise, resources can be provided in an anonymised fashion. This is particularly desirable in industrial Grid usage where the same resources might provide services for competing companies.

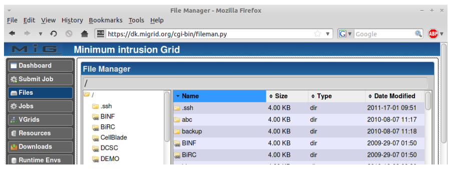
Straightforward Comfortable Usage. In analogy to the system related principle of minimal intrusion, a Grid system should support the user in a straightforward manner. The MiG middleware helps beginners by hiding complex features and providing reasonable defaults for any optional feature.

Advanced Support for Group Collaboration. It has been stated [7] that the essence of Grid is collaboration in virtual organisations, so a Grid middleware should provide appropriate tools especially for this task. MiG provides a wide range of user-controlled collaborative structures and tools, to truly enable collaborating virtual organisations and improve the overall Grid usability.



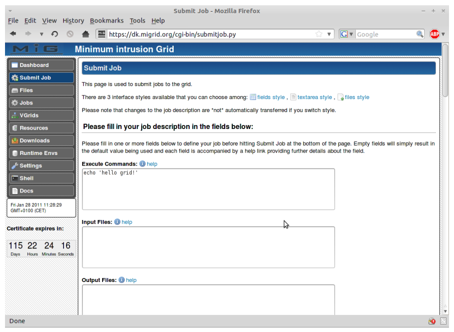
Job ID	Status	Date
334851_1_28_2011__10_34_36_dk.migrid.org.0	QUEUED	Fri Jan 28 10:34:36 2011
334850_1_28_2011__10_33_35_dk.migrid.org.0	FINISHED	Fri Jan 28 10:33:35 2011
334849_1_28_2011__10_31_14_dk.migrid.org.0	FINISHED	Fri Jan 28 10:31:15 2011

(a) Job Management



Name	Size	Type	Date Modified
.ssh	4.00 KB	dir	2011-17-01 09:51
abc	4.00 KB	dir	2010-08-07 11:17
backup	4.00 KB	dir	2010-08-07 11:18
BINF	4.00 KB	dir	2009-29-07 01:50
BiRC	4.00 KB	dir	2009-29-07 01:50

(b) File Browser



This page is used to submit jobs to the grid.

There are 3 interface styles available that you can choose among: fields style , textarea style , files style

Please note that changes to the job description are *not* automatically transferred if you switch style.

Please fill in your job description in the fields below:

Please fill in one or more fields below to define your job before hitting Submit Job at the bottom of the page. Empty fields will simply result in the default value being used and each field is accompanied by a help link providing further details about the field.

Execute Commands:

```
echo 'hello grid!'
```

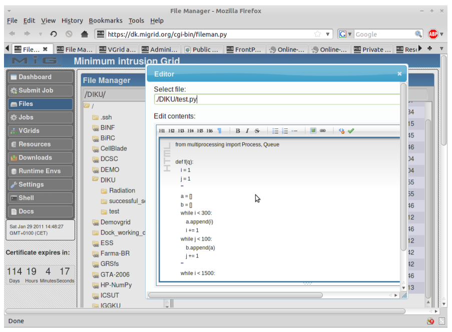
Input Files:

Output Files:

Fri Jan 28 2011 11:28:29 GMT+0100 (CET)

Certificate expires in: 115 Days 22 Hours 24 Minutes 16 Seconds

(c) Job Submission Interface



Select file: ./DIKU/test.py

Edit contents:

```
from multiprocessing import Process, Queue

def f(q):
    i = 1
    j = 1
    '''
    a = []
    b = []
    while i < 300:
        a.append(i)
        i += 1
    while j < 100:
        b.append(a)
        j += 1
    '''
    while i < 1500:
```

Sat Jan 29 2011 14:48:27 GMT+0100 (CET)

Certificate expires in: 114 Days 19 Hours 4 Minutes 17 Seconds

(d) Simple Built-in Editor

Fig. 2. MiG Browser Interface Screenshots

3.3 Browser-Based Interface

The central user interface to a MiG system is the web browser. Users are authenticated and authorised by an X.509 certificate installed in their browser, and can then access all functionality through a single portal. The MiG design carefully avoids transferring any temporary "proxy" credentials to servers or resources.

As Figure 2 illustrates, users should experience MiG as a multiuser workstation with a system account and pre-installed software. The interface to submitting and managing jobs is shown in Figures 2(a) and 2(c). Figures 2(b) and 2(d) demonstrate another essential component of the "workstation" view: the user's home directory on the server and how to manipulate files from within the browser. Some folders are shared with other users (based on virtual organisations), and are seamlessly integrated in the home directory; a custom icon indicates the sharing. We will explain some key features shown in the menu on the left in detail subsequently (VGrids, Resources, and Runtime Env.s).

3.4 How MiG Executes Grid Jobs

Efficient job execution by resources is the primary Grid functionality, realised in MiG in a uniform, straightforward and server-centric manner. Contrary to Globus based Grids, MiG job requests are submitted to a Grid job queue handled by one or more MiG servers from start to finish. There is no direct communication between the user and the resource where the job ends up executing and the user can safely go offline as soon as the job is queued.

Simple Job Specification. Figure 2(c) shows the simple graphical interface with individual fields to specify requirements and actions for a Grid job. While this HTML form is the default interface, users can also directly use the internal job specification language *mRSL*, a simple line-delimited text format. Compared with richer standards such as JSDL or RSL [1, 8], mRSL is simplistic but sufficient. A job specification includes a list of commands to execute, input and output files, and a range of other (optional) properties: hard resource requirements like node and CPU count, soft requirements like memory and disk consumption and wall-clock time limits, and a retry counter for failover scheduling.

Grid Job Scheduling in MiG. Production Grids typically have a long queue of waiting jobs, as there are often more jobs than execution slots. In MiG, pending jobs are stored on servers until a resource becomes available. Resources actively request jobs from a server (*pull* scheduling), and the server selects the best-fitting job according to a configurable scheduling policy. This allows for scheduling jobs across multiple physical resources to optimise throughput.

Scheduling in MiG is guided by the job's software requirements, user-provided memory and disk limits, node and CPU count, and by the execution history of the resource. A number of scheduling algorithms are implemented and deliver good throughput on average while preventing starvation. Apart from the classical variants FIFO, Random and First-Fit, MiG implements a Best-Fit scheduler that avoids occupying oversized resources, a Fair-Fit scheduler which additionally prioritises jobs that have waited longer, and a scheduler based on a Vickrey auction [3], which lays grounds for a pay-per-use "Grid economy" of resources.

3.5 Software Deployment

In order to run anything but trivial jobs on a Grid resource, one typically needs special software. Requirements beyond the basic Grid system have to be be negotiated between users and resources, typically realised in Grid systems by the concept of *Runtime Environments* [12]. A runtime environment in MiG is a data structure that describes system properties – for instance, a special numeric software library, or special hardware – and specifies environment variables to be used in job scripts that require them – for instance, a variable containing necessary compiler flags, the install location, or the path to an executable tool. When a resource provides this library, the owner adds the respective runtime environment to the resource's specification and assigns values to these environment variables. Any MiG user can define runtime environments, but they cannot be modified (only deleted) after creation – redefinitions would cause inconsistencies. And, well-understood, there is no guarantee that any resource in the system implements a particular runtime environment.

Dedicated computing resources usually have their own native software package management system, which only privileged users can use. Users often have to manually intervene and ask a resource owner to install the necessary software, a typical productivity bottleneck in today's production Grids. To address this issue, a mechanism was developed to automatically install software *on-demand* from a software catalogue hosted on the MiG servers. The software catalogue in MiG uses the Zero Install [16] packaging system to support installation and automatic maintenance of pre-packaged software on resources. When a resource provides the ZeroInstall runtime environment, a job can install and use packaged software in a secure and controlled way, instead of using a native installation. Runtime environment specifications are automatically generated from Zero Install package descriptions, making the entire process transparent to the user.

4 MiG Features Beyond the "Job-Shop"

4.1 VGrids: Virtual Organisations in MiG

In early 2000, *Virtual Organisations* (VOs) were stated by Foster [7] as the original "Grid problem", motivation for developing the Grid concept altogether. The term *Virtual Organisation* describes dynamically evolving groups of users that belong to distinct administrative domains, but want to share resources (compute power, storage, software etc.) for a specific purpose. Flexible and dynamic resource sharing across administrative domains is the core of Grid technology.

The MiG system models Virtual Organisations as *VGrids* [11]. As the MiG E/R model in Figure 3 shows, VGrids are hierarchical and act as an organisational entity to define the relationship between resources and users, and among users. Since the model relies on VGrids, it includes a default VGrid which includes all users and where any resource can (but does not have to) participate by default.

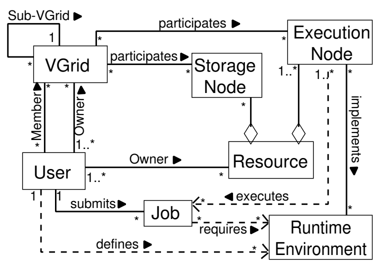


Fig. 3. MiG E/R Model

All services are managed via VGrids: Jobs execute inside a VGrid where the submitter is a member, execution nodes of a resource contribute compute power to members of one or more particular VGrids, and storage nodes expose a specified directory on the resource to members of the VGrids that they have signed up to. Each VGrid also provides a shared folder for members on the server. VGrids are easy to create and manage using the MiG web interface. Furthermore, VGrid operations (adding members or owners, creating sub-structures) are entirely user controlled, enabling ad-hoc VGrid formation and efficient collaboration without administrative hurdles. As we are going to see next, this VGrid concept allows MiG to lift *resource provisioning to user level*, and VGrids can also provide *cloud storage services* and advanced *workflow integration.*

4.2 Resource Management

Any MiG user can add a resource to the Grid through a simple HTML form, by specifying its properties: hardware and software specifications (CPUs, memory, disk size) and login details for the user account that executes the Grid jobs. After submitting the form, the resource is added to the default VGrid.

Different levels of resource trust can thus be modelled with different VGrids. The default VGrid is open to anyone and thus has the lowest level of trust. Resource owners configure which VGrids their resources should participate in as an execution or/and storage resource. Respectively, VGrid owners need to accept the resources into their VGrids before jobs can be executed (to prevent job hijacking). The entire process requires no Grid administrator intervention.

4.3 Storage in MiG

One of the distinctive features of MiG in comparison to other Grid middlewares is the concept of a central user home directory. This is part of the MiG philosophy of the Grid being a virtual workstation: users store their files in their home directory and reference them with relative file names in jobs, and each VGrid provides shared file space which is only visible to VGrid members. The central home directory can be accessed in the MiG web interface through a graphical file manager with context menu and a simple text editor, as shown in Figure 2(d). Optionally, it can even be securely mounted into a user's local file system through an SSHFS interface, to allow transparently working with MiG home files locally.

As useful and intuitive it may be to have files in a centralised MiG home directory, space limitations and privacy policies may prevent users from storing all data there. Jobs can specify external locations for input and output files (via common protocols like HTTP(S), SCP, or (S)FTP), and classified data can be stored in MiG in special VGrid-restricted storage nodes. For each VGrid in which a storage node participates, a directory on the node is securely mounted into the shared VGrid folder in the MiG servers' file system via SSHFS.

4.4 Advanced VGrid features for Group Collaboration

Apart from being a fundamental concept to structure access levels and entities in MiG, VGrids also provide advanced services for group collaboration and resource sharing to their users. MiG servers host shared private folders and classical collaboration software for every VGrid, including public and private web pages, a wiki, a web forum, and a version control system. All this is integrated into the server middleware and browser interface, again in view of the browser being the primary desktop for all activities.

The VGrid web pages and shared folders can be used to realise specific workflows where conceptually similar Grid jobs are submitted frequently and require a custom setup carried out by VGrid owners. As an example, consider a workflow where scientific applications are implemented in a special-purpose language that requires a custom compiler, but the compiled executables can be run on various resources by means of a pre-packaged runtime library. In such a case, a VGrid can be used to provide a dedicated compilation resource, and other resources can use a runtime environment for execution – the whole compilation/execution workflow can be encapsulated in custom HTML forms in the VGrid web space.

We have successfully implemented this workflow encapsulation in two prototypes: one for the McStas neutron raytracing simulator [13] and one for a general Matlab setup using a license-limited Matlab compiler [17]. Both prototypes expose the described workflow of compilation and iterated execution, followed by a post-processing step in case of the McStas software.

To encapsulate the job workflow logics in the VGrid web pages using HTML forms and javascript has clear advantages in usability: no additional authentication is required, and the easy interface for non-experts hides all uninteresting boilerplate code for Grid operation. For the implementor, two other advantages exist: Scientific expert software like the McStas compiler and post-processor is sometimes complicated to set up. In our setup, maintenance can be reduced to only a few dedicated resources that provide the special parts, and the compiled McStas simulation code is ISO C99. The Matlab software setup is easy, but a license is required; in our case only for the dedicated compilation resource.

5 Current Status and Future Directions

MiG started as a proof-of-concept implementation, but became an ongoing success over the last years, thanks to its sage architecture. At the time of writing, our group is operating a MiG installation connecting several compute clusters and special resources based on Cell-based game console and Screen saver software. The system is used for scientific projects in combinatorial genome research, bibliometric analysis, medical imaging, and for teaching purposes. MiG is actively maintained, the latest additions include an improved browser interface, a remote memory library to enable computing resources with limited memory and disk [15], and running virtual machines as interactive Grid jobs.

Development is underway for improving the virtual machine support. One essential ingredient is a light-weight VNC client that runs inside a browser (based on javascript and websockets), to realise the MiG goal of minimal installation requirements. Together with MiG's browser-based interface and the storage resource concept realising cloud storage, the virtual machine support is the final step towards cloud-style IaaS – virtualised resources – embedded in Grid.

6 Conclusions

Grid computing has come of age in the past years, yet its practical incarnations somewhat fall short of the initial vision. We have presented MiG, a Grid middleware that follows rigid principles derived from the original Grid literature and shortcomings observed in existing systems. The stated MiG principles prove useful as a general touchstone for any Grid middleware, and the presentation of MiG demonstrates that such a design is feasible and useful in practice. Especially MiG's advanced features for group collaboration, and in general its VGrid-centric design, are key features for more flexibility and user self-service than usual job-shop Grid systems can provide. The more recent trend of cloud computing has given important impulses to the Grid community, as it advances modern virtualisation techniques and the idea of a consequently consumption-based Grid economy. A principled Grid approach should incorporate modern virtualisation techniques in its services, to realise a strict superset of the cloud vision in terms of stability, user-friendliness, and self-managed virtual collaboration.

Availability. MiG is open source software released under GNU GPL-2. More information and MiG code can be found at `http://www.migrid.org`.

References

1. Anjomshoaa, A., Brisard, F., Drescher, M., Fellows, D., Ly, A., McGough, S., Pulsipher, D., Savva, A.: Job Submission Description Language (JSDL) Specification, V.1.0. Tech. Rep. GFD.136, Grid Forum (2008)
2. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. CACM 53, 50–58 (Apr 2010)
3. Brandt, F., Wei, G.: Vicious strategies for vickrey auctions. In: Müller, J.P., Andre, E., Sen, S., Frasson, C. (eds.) Autonomous Agents 2001, Proceedings. ACM (2001)
4. cURL. Open Source Software, `http://curl.haxx.se/`
5. European Grid Infrastructure, `http://www.egi.eu`
6. Ellert, M., Grønager, M., Konstantinov, A., Kónya, B., Lindemann, J., Livenson, I., Nielsen, J.L., Niinimäki, M., Smirnova, O., Wäänänen, A.: Advanced resource connector middleware for lightweight computational Grids. Future Generation Computer Systems 23, 219–240 (2007)
7. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the Grid: Enabling scalable virtual organizations. Intl. J. of High Perf. Computing Appl. 15(3), 200–222 (2001)
8. Foster, I.: Globus toolkit version 4: Software for service-oriented systems. J. of Computer Science and Technology 21(4), 513–520 (2006)
9. Foster, I., Kesselman, C. (eds.): The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
10. gLite Grid Computing Middleware, `http://glite.web.cern.ch/glite/`
11. Karlsen, H.H., Vinter, B.: VGrids as an Implementation of Virtual Organizations in Grid Computing. In: Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE'06). IEEE Press, New York (2006)
12. Keahey, K., Doering, K., Foster, I.T.: From Sandbox to Playground: Dynamic Virtual Environments in the Grid. In: Buyya, R. (ed.) Grid Computing (GRID 2004), Proceedings. IEEE Press, New York (2004)
13. Lefmann, K., Willendrup, P.: McStas, a Neutron Ray-trace Simulation Package, `http://neutron.risoe.dk`
14. Mell, P., Grance, T.: The NIST Definition of Cloud Computing (Draft). Tech. Rep. 800-145, National Institute of Standards and Technology (NIST) (2011)
15. Rehr, M., Vinter, B.: The User-Level Remote Swap Library. In: High Performance Computing and Communications (HPCC'10). IEEE Press, New York (2010)
16. T. Leonard et al.: Zero Install, a decentralised cross-distribution software installation system. Software under LGPL (2003–2011), `http://www.0install.net`
17. MathWorks: The Matlab Compiler$^{\mathrm{TM}}$, `http://www.mathworks.com/products/compiler`
18. Vinter, B.: The Architecture of the Minimum intrusion Grid (MiG). In: Broenink, J.F., Roebbers, H.W., Sunter, J.P.E., Welch, P.H., Wood, D.C. (eds.) Communicating Process Architectures (CPA'05). IOS Press, Amsterdam (2005)