Architectures, Design Methodologies, and Service Composition Techniques for Grid Job and Resource Management

Per-Olov Östberg

LICENTIATE THESIS, SEPTEMBER 2009
DEPARTMENT OF COMPUTING SCIENCE
UMEÅ UNIVERSITY
SWEDEN

Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden

p-o@cs.umu.se

ISBN 978-91-7264-861-6
ISSN 0348-0542
UMINF 09.15

Printed by Print & Media, Umeå University, 2009

Abstract

The field of Grid computing has in recent years emerged and been established as an enabling technology for a range of computational eScience applications. The use of Grid technology allows researchers and industry experts to address problems too large to efficiently study using conventional computing technology, and enables new applications and collaboration models. Grid computing has today not only introduced new technologies, but also influenced new ways to utilize existing technologies.

This work addresses technical aspects of the current methodology of Grid computing; to leverage highly functional, interconnected, and potentially under-utilized high-end systems to create virtual systems capable of processing problems too large to address using individual (supercomputing) systems. In particular, this thesis studies the job and resource management problem inherent to Grid environments, and aims to contribute to development of more mature job and resource management systems and software development processes. A number of aspects related to Grid job and resource management are here addressed, including software architectures for Grid job management, design methodologies for Grid software development, service composition (and refactorization) techniques for Service-Oriented Grid Architectures, Grid infrastructure and application integration issues, and middleware-independent and transparent techniques to leverage Grid resource capabilities.

The software development model used in this work has been derived from the notion of an ecosystem of Grid components. In this model, a virtual ecosystem is defined by the set of available Grid infrastructure and application components, and ecosystem niches are defined by areas of component functionality. In the Grid ecosystem, applications are constructed through selection and composition of components, and individual components subject to evolution through meritocratic natural selection. Central to the idea of the Grid ecosystem is that mechanisms that promote traits beneficial to survival in the ecosystem, e.g., scalability, integrability, robustness, also influence Grid application and infrastructure adaptability and longevity.

As Grid computing has evolved into a highly interdisciplinary field, current Grid applications are very diverse and utilize computational methodologies from a number of fields. Due to this, and the scale of the problems studied, Grid applications typically place great performance requirements on Grid infrastructures, making Grid infrastructure design and integration challenging tasks. In this work, a model of building on, and abstracting, Grid middlewares has been developed and is outlined in the papers. In addition to the contributions of this thesis, a number of software artefacts, e.g., the Grid Job Management Framework (GJMF), have resulted from this work.

Preface

This thesis consists of a brief introduction to the field, a short discussion of the main problems studied, and the following papers.

Paper I E. Elmroth, P. Gardfjäll, A. Norberg, J. Tordsson, and P-O. Östberg. Designing General, Composable, and Middleware-Independent Grid Infrastructure Tools for Multi-Tiered Job Management. In T. Priol and M. Vaneschi, editors, *Towards Next Generation Grids*, pages 175–184. Springer-Verlag, 2007.

Paper II E. Elmroth, F. Hernández, J. Tordsson, and P-O. Östberg. Designing Service-Based Resource Management Tools for a Healthy Grid Ecosystem. In R. Wyrzykowski et al., editors, *Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, vol. 4967*, pages 259–270. Springer-Verlag, 2008.

Paper III E. Elmroth and P-O. Östberg. Dynamic and Transparent Service Compositions Techniques for Service-Oriented Grid Architectures. In S. Gorlatch, P. Fragopoulou, and T. Priol, editors, *Integrated Research in Grid Computing*, pages 323–334. Crete University Press, 2008.

Paper IV E. Elmroth, S. Holmgren, J. Lindemann, S. Toor, and P-O. Östberg. Empowering a Flexible Application Portal with a SOA-based Grid Job Management Framework. In *The 9th International Workshop on State-of-the-Art in Scientific and Parallel Computing*, to appear, 2009.

Paper V E. Elmroth and P-O. Östberg. A Composable Service-Oriented Architecture for Middleware-Independent and Interoperable Grid Job Management. *UMINF 09.14*, Department of Computing Science, Umeå University, Sweden. Submitted for Journal Publication, 2009.

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

Acknowledgements

A number of people have directly or indirectly contributed to the work in this thesis and deserve acknowledgement. First of all, I would like to thank my advisor Erik Elmroth for not only the opportunities provided and all the hard work, but also for the positive environment he creates in our research group. I would also like to thank my coadvisor Bo Kågström for inspiring discussions and the unique perspective he brings to them. Among my colleagues in the GIRD group I would like to thank Lars Larsson and Johan Tordsson for lengthy discussions of all things more or less related to our work, and Francisco Hernández, Daniel Henriksson, Raphaela Bieber-Bardt, Arvid Norberg, and Peter Gardfjäll (in no particular order) for all their contributions to our collective effort. Among our research partners I would like to thank Sverker Holmgren, Jonas Lindemann, and Salman Toor for interesting collaborations, and the support staff of HPC2N for their contributions and knowledge of the Grid systems we use. Finally, on a personal level I would like to thank my family and friends, without whom none of this would be possible, for all the love and support they provide.
Thank you all.

Contents

Chapter 1

Introduction

In the past decade, Grid computing has emerged and been established as an enabling technology for a range of computational eScience applications. A number of definitions of Grid computing exist, e.g., [33, 37, 64], and while the scientific community has reached a certain level of agreement on what a Grid is [58], best practices for Grid design and construction are still topics for investigation. The definition used in this thesis details Grid computing to be a type of distributed computing focused on aggregation of computational resources for creation of meta-scale virtual supercomputers and systems.

As a paradigm, Grid computing revolves around concepts such as service availability, performance scalability, virtualization of services and resources, and resource (access) transparency [40, 58]. The current methodology of the field is to leverage interconnected high-end systems to create virtual systems capable of great performance scalability, high availability, and collaborative resource sharing [37]. The approach taken in this work employs loosely coupled and decentralized resource aggregation models, assumes resources to be aggregated from multiple ownership domains, and expects all Grid services and components to be subject to resource contention, i.e. to coexist with competing mechanisms.

Grid technology and infrastructure have today found application in fields as diverse as, e.g., life sciences, material sciences, climate studies, astrophysics, and computational chemistry, making Grid computing an interdisciplinary field. Current Grid applications occupy all niches of scientific computation, ranging from embarrassingly parallel high-throughput applications to distributed and synchronized data collation and collaboration projects.

Actors within, and contributions to, the field of Grid computing can broadly be segmented into two main categories; application and infrastructure. Grid applications often stem from independently developed computational methodologies more or less suited for use in Grid environments, and are often limited (in Grid usage scenarios) by how well their methodology lends itself to parallelization. Motivations for migration to Grid environments vary, but often

include envisioned performance benefits or synergetic collaboration effects.

Typically, Grids are designed to provide a level of scalability beyond what is offered by individual supercomputer systems. System requirements vary with Grid application needs, and usually incorporate advanced demands for storage, computational, or transmission capacity, which places great performance requirements on underlying Grid infrastructure at both component and system level. These conditions, combined with typical interdisciplinary requirements of limited end-user system complexity, automation, and high system availability, make Grid infrastructure design and resource federation challenging tasks.

The focus of this thesis lies on research questions related to Grid infrastructure and application design, with emphasis on job and resource management issues. In particular, abstraction of Grid job management interfaces, and related application and infrastructure component integration issues have been studied in the context of federated Grid environments. The methodology of this work includes investigation of architectural design patterns inspired by the notion of an ecosystem of Grid infrastructure components [62], and exploration of Service-Oriented Architecture (SOA)-based [51] implementation techniques. The concept of an ecosystem of Grid infrastructure components, where applications are composed through selection of software from an ecosystem of components, and individual components are subject to meritocratic natural selection, is further described in Section 5.

Two of the overarching goals of the GIRD [63] project, in which this research has been performed, are to investigate and propose architectures for abstraction and provisioning of Grid functionality, and to provide proof-of-concept implementations of proposed architectures. Scientific contributions of this thesis include investigation of architectural design patterns, development of Grid infrastructure task algorithms, and contributions to formulation of design methodologies for scalable Grid infrastructure and application components.

The rest of this thesis is structured as follows. Section 2 provides an introduction to Grid applications and outlines a few of the requirements Grid applications impose on Grid infrastructures and environments. Section 3 discusses Grid infrastructure and covers some of the trade-offs involved in Grid infrastructure design and development. Section 4 provides an overview of the Grid job and resource management problem, and structures the area into constituent processes while briefly referencing some of the work within the field. Section 5 sketches the notion of an ecosystem of Grid components, and serves here as an introduction to the perspective chosen in this work. Section 6 summarizes the contributions of this thesis, and relates the thesis papers to each other. Finally, Section 7 outlines some future directions for this work, and references current efforts related to the work of this thesis.

Chapter 2

Grid Applications

Utilization of Grid technology affords the scientific community to study problems too large to address using conventional computing technology. Use of Grids has resulted in creation of new types of applications and new ways to utilize existing computation-based technology [37]. Grid applications can based on application requirements be segmented into categories such as

- computationally intensive, e.g., interactive simulation efforts such as the SIMRI project [11], and very large-scale simulation and analysis applications such as the Astrophysics Simulation Collaboratory [55].
- data intensive, e.g., experimental data analysis projects such as the European Data Grid [56], and image and sensor analysis applications such as SETI@home [3].
- distributed collaboration efforts, e.g., online instrumentation tools such as ROADnet [45], and remote visualization projects such as the Virtual Observatory [66].

Based on computational requirements and topology of the application, computational Grid applications can broadly be classified as High Performance Computing (HPC), High Throughput Computing (HTC), or hybrid approaches. HPC applications are generally concerned with system peak performance, and measure efficiency in the amount of computation performed on dedicated resource sets within limited time frames. Computations are in HPC applications typically structured to maximize application computational efficiency for a particular problem, e.g., through Message Passing Interface (MPI) [57] frameworks. HTC applications are conversely focused on resource utilization and measure performance in the amount of computation performed on shared resource sets over extended periods of time, e.g., in tasks per month. Computationally, HTC applications are generally composed of large numbers of (small) independent jobs running on non-dedicated resource sets without real-time constraints for result delivery. A number of hybrids between the HPC and HTC

paradigms exist, e.g., the more recently formulated Many Task Computing (MTC) [54] paradigm. MTC applications focus on running large amounts of tasks over short periods of time, are typically communication-intensive but not naturally expressed using synchronized communication patterns like MPI, and measure performance using (application) domain-specific metrics.

Beside obvious computational requirements, Grid applications typically also impose advanced system performance requirements for, e.g.,

- storage capacity: Grid applications potentially process very large data sets, and often do so without predictable access patterns.
- data transfer capabilities: Grid computations are typically brokered and may be performed far from the original location for input data and application software. Efficient data transfer mechanisms are required to relocate data to computational resources, and return results after computation.
- usability: Grid interfaces abstract resource system complexity and use of underlying computational resources to improve system usability and lower learning requirements.
- scalability: Grid application system requirements are likely to vary during application runtime, requiring underlying systems and infrastructure to access and scale computational, storage, and transfer capabilities on demand.
- availability: Grid applications and systems are typically composed through aggregation of computational resources, allowing Grids to exhibit very high levels of system availability despite system capacity varying over time due to resource volatility issues. Consistent levels of system access and quality of service improve the perception of Grid availability and stability.
- collaboration: Grid applications and systems support levels of collaboration ranging from multiple users working on shared data to multiple organizations utilizing shared resources.

System complexity and the great demands and different requirements of current Grid applications have led to the emergence of two major types of Grids; computational Grids and data Grids. Typically, computational Grids focus on providing abstracted views of computational resource access, and address very large computational problems. Data Grids conversely focus on providing virtualization of data storage capabilities and provide non-trivial and scalable qualities of service for very large data sets. A number of additional Grid system subtypes have also emerged, e.g., collaboration Grids, enterprise Grids, and cluster Grids [58]. The work in this thesis has been focused on systems designed for use in computational Grid environments.

From a performance perspective, the construction of Grid systems is facilitated by improvements in computational and network capacity, and motivated by general availability of highly functional and well connected end systems. Increase in network capacity alone has lead to changes in computing geometry and geography [37], and technology advances have today made massive-scale collaborative resource sharing not only feasible, but approaching ubiquitous.

From an application perspective, Grid computing holds promise of more efficient models for collaboration when addressing larger and more complex problems, less steep learning curves (as compared to traditional high-performance computing), increased system utilization rates, and efficient computation support for broader ranges of applications. While Grids of today have achieved much in system utilization, scalability and performance, much work in reducing system complexity and increasing system usability still remain [58].

Chapter 3

Grid Infrastructure

The name Grid computing originated from an analogy in the initial guiding vision of the field; to provide access to the capabilities of computational resources in a way similar to how power grids provide electricity [37], i.e. with transparency in

- resource selection (i.e. which resource to use).
- resource location (i.e. with transparency in resource access).
- resource utilization (i.e. amount of resource capacity used).
- payment models (i.e. pay for resource utilization rather than acquisition).

In this vision, the role of Grid infrastructure becomes similar to that of power production infrastructure: to provide capacity to systems and end-users in cost-efficient, transparent, federated, flexible, and accessible manners. While application and user requirements on Grids vary greatly, and can be argued to be more complex than those of power infrastructure, the analogy is apt in describing a federated infrastructure providing flexible resource utilization models and consistent qualities of service through well-defined interfaces.

To realize a generic computational infrastructure capable of flexible utilization models, it is rational to build on standardized, reusable components. The approach of this work is to identify and isolate well-defined Grid functionality sets, and to design interfaces and architectures for these in manners that allow components to be used as building blocks in construction of interoperable Grid applications and systems [24, 26]. In an analogy to efforts within related fields, the role of generic Grid components could be compared to the role of, e.g., frameworks such as Linear Algebra PACKage (LAPACK) [5] libraries in numerical linear algebra. Similarly, the role of application integration components could be compared to the role of Basic Local Alignment Search (BLAST) [60] toolkits in bioinformatics, and component interfaces to Basic Linear Algebra Subprograms (BLAS) [12] Application Programmer Interfaces (APIs).

From a systems perspective, Grid computing revolves around concepts such as (performance) scalability, virtualization, and transparency [40]. Performance scalability here refers to the ability of a system to dynamically increase the computational (or storage, network, etc.) capacity of the system to meet the requirements of an application on demand. Virtualization here denotes the process of abstracting computational resources, a practice that can be found on all levels of a Grid. For example, Grid applications' use of infrastructure is often abstracted and hidden from end-users, Grid systems and infrastructure typically abstract the use of computational resources from the view of applications, and access to Grid computational resources is typically abstracted by native resource access layers, e.g., batch systems. The term transparency is used to describe that, like access to systems and system components, scalability should be automatic and not require manual efforts or knowledge of underlying systems to realize access to, or increase in, system capacity.

Typically today, performance scalability is achieved in Grid systems through dynamic provisioning of multiple computational resources over a network, virtualization through interface abstraction mechanisms, and transparency through automation of core Grid component tasks (such as resource discovery, resource brokering, file staging, etc.).

To facilitate flexible resource usage models, Grid users and resource allotments are typically organized in Virtual Organizations (VOs) [38]. VOs is a key concept in Grid computing that pertains to virtualization of a system's user base around a set of resource-sharing rules and conditions. The formulation of VOs stems from the dynamical nature of resource sharing where resource availability, sharing conditions, and organizational memberships vary over time. This mechanism allows Grid resource usage allotments to be administrated and provided by decentralized organizations, to whom individual users and projects can apply for memberships and resource usage credits. VOs employ scalable resource allotment mechanisms suitable for cross-ownership domain aggregation of resources, and provide a way to provision resource usage without pre-existing trust relationships between resource owners and individual Grid users.

In summary, a Grid computing infrastructure should provide flexible and secure resource access and utilization through coordinated resource sharing models to dynamic collections of individuals and organizations. Furthermore, resources and users should be organized in Virtual Organizations and systems be devoid of centralized control, scheduling omniscience, and pre-existing trust relationships.

Chapter 4

Grid Job and Resource Management

A core task set of any Grid infrastructure is job and resource management, a term here used to collectively reference a set of processes and issues related to execution of programs on computational resources in Grid environments. This includes, e.g., management, monitoring, and brokering of computational resources; description, submission, and monitoring of jobs; fairshare scheduling [46] and accounting in Virtual Organizations; and various cross-site administrational and security issues.

Grid job and resource management tasks seem intuitive when viewed individually, but quickly become complex when considered as parts of larger systems. A number of component design trade-offs, requirements, and conditions are introduced by core Grid requirements for, e.g., system scalability and transparency, and tend to become oxymoronic when individual component designs are kept strictly task oriented. An approach taken in this work is to primarily regard components as parts of systems, and focus on component interoperability to promote system composition flexibility [26]. The primary focus of the Grid job and resource management contributions here is to abstract system complexity and heterogeneity, and to allow applications to leverage resource capabilities without becoming tightly coupled to particular Grids or Grid middlewares [27].

4.1 Grid Environments

Grid systems are composed through aggregation of multiple cooperating computing systems, and federated Grid environments are realized through (possibly hierarchical) federation of existing Grids.

In the naive model illustrated in Figure 1, regional organizations aggregate dedicated cluster-based resources from local supercomputing centers to form

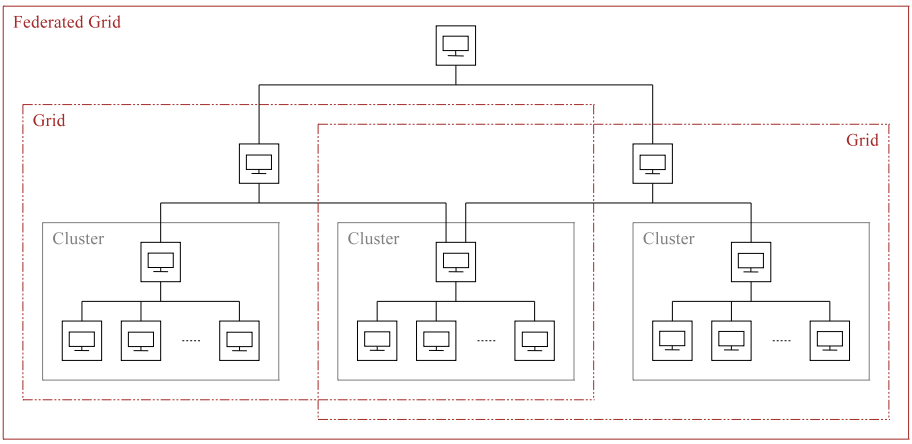

Figure 1: A naive Grid model. Grids aggregate clusters of computational resources, which may be part of multiple Grids. Federated Grid environments are composed from collaborative federation of existing Grids.

computational Grids. Due to the relatively homogeneous nature of today's supercomputers, such Grids typically exhibit low levels of system heterogeneity, and administrators can to a large extent influence system configuration and resource availability.

As also illustrated in Figure 1, international Grids are typically formed from collaborative federation of regional, and other existing Grids. As federated Grids typically aggregate resources from multiple Grids and resource sites, a natural consequence of resource and Grid federation is an increased degree of system heterogeneity. System heterogeneity may be expressed in many ways, e.g., through heterogeneity in hardware and software, resource availability, accessibility, and configuration, as well as in administration policies and utilization pricing. Technical heterogeneity issues are in Grid systems addressed through interface abstraction methods and generic resource description techniques, which allow virtualization of underlying resources and systems.

A core requirement in Grid systems is that resource owners at all levels retain full administrative control over their respective systems. This Grid characteristic, to be devoid of centralized control [33], is a design trait aimed to promote scalability in design and implementation of Grids, and imposes a number of cross-border administrational and security issues.

Security issues naturally arise in federation of computational resources over publicly accessible networks, i.e. the Internet, and are in Grid infrastructures addressed through use of strong cryptographic techniques such as Public Key Infrastructures (PKI) [49]. Grid users are typically organized in VOs, which stipulate rules and conditions for access to Grid resources, and authenticated through established security mechanisms such as PKI certificates [1].

4.2 Grid Resources and Middlewares

A typical HPC Grid resource consists of a high-end computer system equipped with (possibly customized) software such as

- data access and transfer utilities, e.g., GridFTP [18].
- batch systems and scheduling mechanisms, e.g., PBS [10] and Maui [48].
- job and resource monitoring tools, e.g., GridLab Mercury Monitor [8].
- computation frameworks, e.g., BLAST [60].

HTC resources are of more varied nature, CPU-cycle scavenging schemes such as Condor [61] for example typically utilize standard desktop machines, while volunteer computing efforts such as distributed.net [17] may see use of any type of computational resource provided by end-users. HTC Grids often deploy softwares that can be considered part of Grid middlewares on computational resources, e.g., Condor and BOINC [2] clients.

Grids are created through aggregation of computational resources, typically using Grid middlewares to abstract complexity and details of native resource systems such as schedulers and batch systems. Grid middlewares are (typically distributed) systems that act on top of local resource systems, abstracting native system interfaces, and provide interoperability between computational systems. To applications, Grid middlewares offer virtualized access to resource capabilities through abstractive job submission and control interfaces, information systems, and authentication mechanisms.

A number of different Grid middlewares exist, e.g., ARC [19], Globus [42], UNICORE [59], LCG/gLite [13], and vary greatly in design and implementation. In a simplified model, Grid middlewares contain functionality for

- resource discovery, often through specialized information systems.
- job submission, monitoring, and management.
- authentication and authorization of users.

Additionally, middlewares and related systems can incorporate solutions for advanced functionality such as resource brokering [61], accounting [41], and Grid-wide load balancing [13].

While one of the original motivations for construction of Grids where to address resource heterogeneity issues, complexity and size of Grid middlewares have led to a range of middleware interoperability issues, and given rise to the Grid interoperability contradiction [30]; Grid middlewares are not interoperable, and Grid applications are not portable between Grids. The Grid interoperability contradiction results in Grid applications being tightly coupled to Grid middlewares, and a lack of generic tools for Grid job management.

4.3 Resource Management

Grid resources are typically owned, operated, and maintained by local resource owners. Local resource sharing policies override Grid resource policies; computational resources shared in Grid environments according to defined schedules are possibly not available to Grid users outside scheduled hours. Due to this, and hardware and software failures, administrational downtime, etc., Grid resources are generally considered volatile.

In Grid systems, resource volatility is typically abstracted using dynamic service description and discovery techniques, utilizing loosely coupled models [65] for client-resource interaction. Local resource owners publish information about systems and resources in information systems, and Grid clients, e.g., resource brokers and submission engines, discover resources on demand and utilize the best resources currently available during the job submission phase.

Reliable resource monitoring mechanisms are critical to operation in Grid environments. While resource characteristics, e.g., hardware specifications and software installations, can be considered static, factors such as resource availability, load, and queue status are inherently dynamic. To facilitate Grid utilization and resource brokering, resource monitoring systems are used to provide information systems resource availability and status data.

As resource monitoring systems and information systems in Grid environments typically exist in different administrational domains, resource status information need to be disseminated through well-defined, machine-interpretable interfaces. The Web Service Resource Framework (WSRF) [35] specification family addresses Web Service state management issues, and contain interface definitions and notification mechanisms suitable for this task. In Grid environments, information systems potentially contain large quantities of information and can be segmented and hierarchically aggregated to partition resource information into manageable proportions.

4.4 Resource Brokering

A fundamental task in Grid job management is resource brokering; matching of a job to computational resource(s) suitable for job execution. In this model, resource brokers operate on top of Grid middlewares, and rely on information systems and job control systems to enact job executions.

Typically in Grid resource brokering, jobs are represented by job descriptions, which contain machine-readable representations of job characteristics and job execution meta-data. A number of proposed job description formats exist, including middleware-specific solutions such as Globus RSL [34], ARC XRSL [19], and standardization efforts such as JSDL [6]. Job descriptions provide information such as

- program to execute.

- parameters and environmental settings.
- hardware requirements, e.g., CPU, storage, and memory requirements.
- software requirements, e.g., required libraries and licenses.
- file staging information, e.g., data location and access protocols.
- meta-information, e.g., duration estimates and brokering preferences.

Resource brokering is subject to heuristic constraints and optimality criteria such as minimization of cost, maximization of resource computational capacity, minimization of data transfer time, etc., and is typically complicated by factors such as missing or incomplete brokering information, propagation latencies in information systems, and existence of competing scheduling mechanisms [30].

A common federated Grid environment characteristic designed to promote scalability is absence of scheduling omniscience. From this, two fundamental observations can be made. First, no scheduling mechanism can expect to monopolize job scheduling, all schedulers are forced to collaborate and compete with other mechanisms. Second, due to factors such as system latencies, information caching and status polling intervals, all Grid schedulers operate on information which to some extent is obsolete [31]. In these settings, Grid brokers and schedulers need to adapt to their environments and design emphasis should be placed on coexistence [27]. In particular, care should be taken to not reduce total Grid system performance, or performance of competing systems, through inefficient mechanisms in brokering and scheduling processes.

4.5 Job Control

Once resource brokering has been performed, and rendered a suitable computational resource candidate set, jobs can be submitted to resources for execution. For reasons of virtualization and separation of concerns, this is typically done through Grid middleware interfaces rather than directly to native resource interfaces, as resource heterogeneity issues would needlessly complicate clients and applications. Normally, execution of a Grid job on a computational resource adheres to the following schematic.

1. submission: job execution time is allocated at the resource site, i.e. the job is submitted to a resource job execution queue.
2. stage in: job data, including data files, scripts, libraries, and executables required for job execution are transfered to the computational resource as specified by the job description.
3. execution: the job is executed and monitored at the resource.
4. stage out: job data and result files are transfered from the computational resource as specified by the job description.

5. clean up: job data, temporary, and execution files are removed from the computational resource.

Naturally, ability to prematurely abort and externally monitor job executions must be provided by job control systems. In general, most systems of this complexity are built in layers, and Grid middlewares typically provide job control interfaces that abstract native resource system complexity.

As in any distributed system, a number of failures ranging from submission and execution failures to security credential validation and file transfer errors may occur during the job execution process. To facilitate client failure management and error recovery, failure context information must be provided clients. In Grid systems, failure management is complicated by factors such as resource ownership boundaries and resource volatility issues. Care must also be taken to isolate jobs executions, and to ensure that distribution of failure contexts not result in information leakage. Typically, Grids make use of advanced security features that make failure management, administration, and direct access to resource systems complex.

4.6 Job Management

Beyond generic resource brokering and job control capabilities, there exists a functionality set required by advanced high-level Grid applications. For example, efficient mechanisms for monitoring and workflow-based scheduling of jobs can greatly facilitate management of large sets of jobs.

Two types of Grid job monitoring mechanisms exist, pull-based and push-based. In pull models, clients and brokers poll resource status to detect and respond to changes in job and resource status. As jobs and Grid clients typically outnumber available Grid resources, polling-based resource update models scale poorly. As clients and resources exist in different ownership domains, pull models are also sometimes considered intrusive.

In push models, Grid resources, or systems monitoring them, publish status updates for jobs and resources in information systems or directly to interested clients. Push updates typically employ publish-subscribe communication patterns, where interested parties register for updates in advance, e.g., during job submission. In Grid systems, push models provide several performance benefits compared to pull models. Push models improve system scalability through reduced system load and decreased communication volumes, and may sometimes simplify client-side system design as they afford clients to act reactively rather than proactively. This reduced client complexity comes at the cost of increased service-side complexity. As Grid resources are volatile, systems distributed, and most Grids employ unreliable communication channels, push models must often be supplemented with pull model mechanisms. Push notifications can also be extended to notification brokering scenarios, and be incorporated in Message-Oriented Middleware (MOM) [9] or Enterprise Service Bus (ESB) [14]-like notification brokering schemes. The WS-Notification

[43] details interfaces for push model status notifications suitable for Grid job management architectures.

A common advanced Grid application requirement is to, possibly conditionally, run batches of jobs sequentially or in parallel. One way to organize these sets is in Grid workflows [50], where job interdependencies and coordination information are expressed along with job descriptions. In simple versions, workflows can be seen as job descriptions for sets of jobs. In more advanced versions, e.g., the Business Process Execution Language (BPEL) [4], workflows may themselves contain script-like instruction sets for, e.g., conditional execution, looping, and branching of jobs. When using workflows, Grid applications rely on workflow engines, e.g., Taverna [52], Pegasus [16], and Grid infrastructures to automate execution of job sets. An important question here becomes abstraction of level of detail, and balancing of level of detail against level of control for advanced job management systems [23].

Advanced job management systems may also provision functionality for customization of job execution, control, and management. In this case, job management components should provide interfaces for customization that does not require end-users or administrators to replace entire system components, but rather offer flexible configuration and code injection mechanisms [26, 27].

4.7 (Non-Intrusive) Interoperability

A large portion of Grid infrastructure operation builds on automation of Grid functionality tasks. This is achieved through Grid component and system collaboration, and thus require systems participating in Grids to provide machine-interpretable and interoperable system interfaces. Due to Grid heterogeneity issues stemming from Grid and resource federation, properties such as platform, language, and versioning independence become highly desirable. For these reasons, Grid components typically build on open standards and formats, and utilize technologies that facilitate system interoperation, e.g., XML and Web Services [40]. To promote non-intrusive interoperability in Grid system design, many Grid systems are realized as Service-Oriented Architectures [51].

Grid standardization efforts have proposed interfaces for many interoperability systems ranging from job description formats, e.g., JSDL [6], job submission and control interfaces, e.g., OGSA BES [36], to resource discovery, e.g., OGSA RSS [39], and Cloud computing interfaces, but broad consensus on best practices for Grid construction has yet to be reached.

Chapter 5

An Ecosystem of Grid Components

Currently, a number of open research questions regarding Grid and Cloud computing software design are being addressed by the scientific community. A common problem in current efforts is that applications tend to be tightly coupled to specific middlewares or Grids, and lack ability to be generally applicable to computational problems [25]. This work addresses Grid software design methodologies for computational eScience applications that support the majority of current computational approaches, and places focus on infrastructure composition and scalability rather than specific problem sets [24].

The methodology of this work builds on the idea of an ecosystem of Grid infrastructure components [62], which encompasses a view of a software ecosystem where individual components compete and collaborate for survival on an evolutionary basis. Fundamental to this idea is the notion of software niches, areas of functionality defined and populated by software components that interact and provision use of Grid resources to applications and end-users. Here, standardization of interfaces and software components help define niche boundaries, and continuous development of Grid infrastructure components and integration with eScience applications help shape and redefine niches (as well as the ecosystem at large) through competition, innovation, diversity, and evolution.

In this approach, identification and exploration of component and system traits likely to promote software survival in the Grid ecosystem are central, and generally help in identification and formulation of research questions. Softwares designed using this methodology focus on establishment of core functionality, and adapt to, and integrate with, members of neighboring niches rather than attempt to replace them.

Currently, advanced eScience applications and computational infrastructures require software and systems to scale with problem complexity and simultaneously abstract heterogeneity issues introduced by this scalability. For

usability, software also require interoperability and robustness to enable automation of repetitive tasks in computational environments, and flexibility in configuration and deployment to be employed in environments with great variance in usage and deployment requirements. The approach taken in this work is to build on top of Grid middlewares and create layers of flexible software that interoperate non-intrusively with components from different niches in the Grid ecosystem, and allow applications to be decoupled from Grid middlewares.

Chapter 6

Thesis Contributions

Large portions of the work in this thesis focus on Grid job and resource management issues, and address how these can be approached using middleware-independent techniques. Two of the papers outline and discuss approaches to Grid software development, one from a software engineering perspective (II), and one from a system (re)factorization point of view (III). Two of the papers (I and V) investigate and outline a generic architecture for Grid job management capable of adoption in a majority of existing Grid computing environments. Paper IV studies integration issues related to use of the proposed job management architecture, and details an integration architecture building on it.

6.1 Paper I

Paper I [21] investigates software design issues for Grid job management tools. Building on experiences from previous work [28, 29, 31], an architectural model for construction of a middleware-independent Grid job management system is proposed, and the design is detailed from an architectural point of view. In this work, a layered architecture of composable services that each manage a separate part of the Grid job management process is outlined, and design and implementation implications of this architecture are discussed. The architecture separates applications from infrastructure through a customizable set of services, and provides middleware-independence through use of (possible third party) middleware adaption plug-ins.

A Globus Toolkit 4-based [34] prototype implementation of some of the services in the architecture is presented, and the services are integrated with the ARC [19] and Globus [42] middlewares. To demonstrate the feasibility of the approach, preliminary results from prototype testing are presented along with an evaluation of system performance and system use cases.

6.2 Paper II

Paper II [24] analyzes Grid software development practices from a software engineering perspective. An approach to software development for high-level Grid resource management tools is presented, and the approach is illustrated by a discussion of software engineering attributes such as design heuristics, design patterns, and quality attributes for Grid software development.

The notion of an ecosystem of Grid infrastructure components is extended upon, and Grid component coexistence, composability, adoptability, adaptability, and interoperability are discussed in this context. The approach is illustrated by five case studies from recent software development efforts within the GIRD project; the Job Submission Service (JSS) [31], the Grid Job Management Framework (GJMF) [27], the Grid Workflow Execution Engine (GWEE) [22], the SweGrid Accounting System (SGAS) [41], and the Grid-Wide Fairshare Scheduling System (FSGrid) [20].

6.3 Paper III

Paper III [26] investigates Service-Oriented Architecture-based techniques for construction of Grid software, and details a set of service composition techniques for use in Grid infrastructure environments. Transparent service decomposition and dynamic service recomposition techniques are discussed in a Grid software (re)factorization setting, and implications of their use are elaborated upon. A set of architectural design patterns and service development mechanisms for service refactorization, service invocation optimization, customization of service mechanics, dynamic service configuration, and service monitoring are presented in detail, and synergetic effects between the patterns are discussed. Examples of use of the patterns in actual software development efforts are used throughout the paper to illustrate the presented approach.

6.4 Paper IV

Paper IV [25] addresses Grid software integration issues and discusses problems inherent to Grid applications being tightly coupled to Gird middlewares. The paper proposes an architecture for system integration focused on seamless integration of applications and Grid middlewares through a mediating layer handling resource brokering and notification delivery. The proposed architecture is illustrated in a case study where the LUNARC application portal [47] is integrated with the Grid Job Management Framework [27] presented in papers I and V. The proposed integration architecture is evaluated in a performance evaluation and findings from the integration efforts are presented throughout the paper.

6.5 Paper V

Paper V [27] further elaborates on the work of Paper I, and proposes a composable Service-Oriented Architecture-based framework architecture for middleware-independent Grid job management. The proposed architecture is presented in the context of development and deployment in an ecosystem of Grid components, and software requirements and framework composition are discussed in detail. The model of Paper I is extended with additional services for job description translation, system monitoring and logging, as well as a broader integration support functionality range. Furthermore, a proof-of-concept implementation of the entire framework is presented and evaluated in a performance evaluation that illustrates some of the major trade-offs in framework use.

The Grid ecosystem model of Paper II is further developed and discussed in the context of the proposed job management architecture, and the software composition techniques of Paper III are built upon and evaluated in the context of this project. Throughout the paper, a number of software design and implementation findings are presented, and the framework is related to a set of similar software development efforts within adjoining Grid ecosystem niches.

Chapter 7

Future Work

A number of possible future extensions to the work of this thesis have been identified, some of which are currently pursued within the GIRD project. Further development, and documentation of experiences from use of the software development model of Paper II is a continuous effort, and of current special interest is adoption of the model to Cloud computing software development efforts. The model itself is currently utilized in a number of projects under the GIRD multi-project umbrella, and are in the projects of this thesis combined with the techniques of Paper III.

The service composition techniques of Paper III have been further developed in work on Paper V, and are currently under investigation for extension in a code-generation effort within multiple projects. The techniques lend themselves well to software refactorization efforts and prototype implementations are being developed for integration with the Apache Axis2 [7] SOAP [44] engine. Extension of these techniques to Representational State Transfer (REST)-based [32] Resource-Oriented Architectures (ROA) [53] would possibly be a viable alternative to current Web Service Description Language (WSDL)-based [15] code generation. In this case the abstraction of the mechanisms would naturally be placed in API implementations, instead of in generated stub code. Extension of these techniques to a more ubiquitous notification scheme, where the current WSRF-based [35] approach could be extended to a more generic MOM- [9] or ESB-based [14] approach would also be possible. Development of a more generic framework for service development adapted to a larger number of service engines would further such efforts.

The job management framework of Paper I and V is currently being developed into a more mature software product scheduled for use in SweGrid, the Swedish national Grid, and a port of the framework to alternative SOAP stacks is currently under investigation. Interesting research questions related to the architecture of this framework include, e.g., development of data management capabilities, (further) adaption to standardization efforts, investigation of advanced notification brokering capabilities, and inclusion of advanced resource

brokering features such as advance reservation and coallocation of resources, and classadd-based match-making.

Further development and integration of high-level job clients such as workflow engines and Grid portals would be beneficial, as well as further investigation of integration architectures such as that of Paper IV, as these are expected to increase the understanding of application-infrastructure integration issues. Investigation of (minimalistic) implementation approaches for Grid middleware development and simulation are also expected to render a deeper understanding of these issues. Integration with Cloud Computing solutions, and other virtualization-based infrastructure techniques, are also of interest and can be expected to increase the adoptability and flexibility of these techniques.

Bibliography

[1] C. Adams and S. Farrell. Internet X. 509 public key infrastructure certificate management protocols, 1999.

[2] D.P. Anderson. BOINC: A system for public-resource computing and storage. In *5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, 2004.

[3] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@ home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[4] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, et al. Business process execution language for web services, version 1.1. *Specification, BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems*, 2003.

[5] E. Angerson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammarling, J. Demmel, C. Bischof, and D. Sorensen. LAPACK: A portable linear algebra library for high-performancecomputers. In *Proceedings of Supercomputing'90*, pages 2–11, 1990.

[6] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, A. S. McGough, D. Pulsipher, and A. Savva. Job Submission Description Language (JSDL) specification, version 1.0. http://www.ogf.org/documents/GFD.56.pdf, September 2009.

[7] Apache Web Services Project - Axis2. http://ws.apache.org/axis2, September 2009.

[8] Z. Balaton and G. Gombas. Resource and job monitoring in the grid. *Lecture notes in computer science*, pages 404–411, 2003.

[9] G. Banavar, T. D. Chandra, R. E. Strom, and D. C. Sturman. A case for message oriented middleware. In *Proceedings of the 13th International Symposium on Distributed Computing*, pages 1–18, London, UK, 1999. Springer-Verlag.

[10] A. Bayucan, R.L. Henderson, C. Lesiak, B. Mann, T. Proett, and D. Tweten. Portable Batch System: External reference specification. Technical report, Technical report, MRJ Technology Solutions, 1999.

[11] H. Benoit-Cattin, G. Collewet, B. Belaroussi, H. Saint-Jalmes, and C. Odet. The SIMRI project: a versatile and interactive MRI simulator. *Journal of Magnetic Resonance*, 173(1):97–115, 2005.

[12] BLAS (Basic Linear Algebra Subprograms). http://www.netlib.org/blas/. September 2009.

[13] J. Knobloch (Chair) and L. Robertson (Project Leader). LHC computing Grid technical design report. http://lcg.web.cern.ch/LCG/tdr/, September 2009.

[14] D. Chappell. *Enterprise Service Bus*. O'Reilly Media, Inc., 2004.

[15] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, September 2009.

[16] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, A. Laity, J.C. Jacob, and D.S. Katz. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[17] distributed.net. http://www.distributed.net/. September 2009.

[18] W. Allcock (editor). GridFTP: Protocol extensions to FTP for the Grid. http://www.ogf.org/documents/GFD.20.pdf, September 2009.

[19] M. Ellert, M. Grønager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J. L. Nielsen, M. Niinimäki, O. Smirnova, and A. Wäänänen. Advanced resource connector middleware for lightweight computational Grids. *Future Generation Computer Systems. The International Journal of Grid Computing: Theory, Methods and Applications*, 27(2):219–240, 2007.

[20] E. Elmroth and P. Gardfjäll. Design and evaluation of a decentralized system for Grid-wide fairshare scheduling. In H. Stockinger, R. Buyya, and R. Perrott, editors, *e-Science 2005, First International Conference on e-Science and Grid Computing*, pages 221–229. IEEE CS Press, 2005.

[21] E. Elmroth, P. Gardfjäll, A. Norberg, J. Tordsson, and P-O. Östberg. Designing General, Composable, and Middleware-Independent Grid Infrastructure Tools for Multi-Tiered Job Management. In T. Priol and M. Vaneschi, editors, *Towards Next Generation Grids*, pages 175–184. Springer-Verlag, 2007.

[22] E. Elmroth, F. Hernández, and J. Tordsson. A light-weight Grid workflow execution engine enabling client and middleware independence. In R. Wyrzykowski et al., editors, *Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, vol. 4967*, pages 754–761. Springer-Verlag, 2008.

[23] E. Elmroth, F. Hernández, and J. Tordsson. Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment. *Future Generation Computer Systems. The International Journal of Grid Computing: Theory, Methods and Applications*, 2009, to appear.

[24] E. Elmroth, F. Hernández, J. Tordsson, and P-O. Östberg. Designing Service-Based Resource Management Tools for a Healthy Grid Ecosystem. In R. Wyrzykowski et al., editors, *Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, vol. 4967*, pages 259–270. Springer-Verlag, 2008.

[25] E. Elmroth, S. Holmgren, J. Lindemann, S. Toor, and P-O. Östberg. Empowering a Flexible Application Portal with a SOA-based Grid Job Management Framework. In *The 9th International Workshop on State-of-the-Art in Scientific and Parallel Computing*, to appear, 2009.

[26] E. Elmroth and P-O. Östberg. Dynamic and Transparent Service Compositions Techniques for Service-Oriented Grid Architectures. In S. Gorlatch, P. Fragopoulou, and T. Priol, editors, *Integrated Research in Grid Computing*, pages 323–334. Crete University Press, 2008.

[27] E. Elmroth and P-O. Östberg. A Composable Service-Oriented Architecture for Middleware-Independent and Interoperable Grid Job Management. *UMINF 09.14*, Department of Computing Science, Umeå University, Sweden. Submitted for Journal Publication, 2009.

[28] E. Elmroth and J. Tordsson. An interoperable, standards-based Grid resource broker and job submission service. In H. Stockinger, R. Buyya, and R. Perrott, editors, *e-Science 2005, First International Conference on e-Science and Grid Computing*, pages 212–220. IEEE CS Press, 2005.

[29] E. Elmroth and J. Tordsson. A Grid resource broker supporting advance reservations and benchmark-based resource selection. In J. Dongarra, K. Madsen, and J. Waśniewski, editors, *Applied Parallel Computing - State of the Art in Scientific Computing, Lecture Notes in Computer Science vol. 3732*, pages 1061–1070. Springer-Verlag, 2006.

[30] E. Elmroth and J. Tordsson. Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions. *Future Generation Computer Systems. The International Journal of Grid Computing: Theory, Methods and Applications*, 24(6):585–593, 2008.

[31] E. Elmroth and J. Tordsson. A standards-based grid resource brokering service supporting advance reservations, coallocation and cross-grid interoperability. *Concurrency Computat.: Pract. Exper.*, 2009. accepted.

[32] R. T. Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[33] I. Foster. What is the grid? a three point checklist. *GRID today*, 1(6):22–25, 2002.

[34] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In H. Jin, D. Reed, and W. Jiang, editors, *IFIP International Conference on Network and Parallel Computing, LNCS 3779*, pages 2–13. Springer-Verlag, 2005.

[35] I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, and S. Weerawarana. Modeling stateful resources with Web services. http://www-106.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf, September 2009.

[36] I. Foster, A. Grimshaw, P. Lane, W. Lee, M. Morgan, S. Newhouse, S. Pickles, D. Pulsipher, C. Smith, and M. Theimer. OGSA© basic execution service version 1.0. http://www.ogf.org/documents/GFD.108.pdf, September 2009.

[37] I. Foster and C. Kesselman. *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann, 2004.

[38] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200, 2001.

[39] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. Von Reich. The Open Grid Services Architecture, version 1.5, 2006. http://www.ogf.org/documents/GFD.80.pdf, May 2009.

[40] I. Foster and S. Tuecke. Describing the elephant: The different faces of IT as service. *ACM Queue*, 3(6):26–34, 2005.

[41] P. Gardfjäll, E. Elmroth, L. Johnsson, O. Mulmo, and T. Sandholm. Scalable Grid-wide capacity allocation with the SweGrid Accounting System (SGAS). *Concurrency Computat.: Pract. Exper.*, 20(18):2089–2122, 2008.

[42] Globus. http://www.globus.org. September 2009.

[43] S. Graham, D. Hull, and B. Murray. Web Services Base Notification 1.3 (WS-BaseNotification). http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf, September 2009.

[44] M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, H. Frystyk Nielsen, A. Karmarkar, and Y. Lafon. SOAP version 1.2 part 1: Messaging framework. http://www.w3.org/TR/soap12-part1/, September 2009.

[45] T. Hansen, S. Tilak, S. Foley, K. Lindquist, F. Vernon, A. Rajasekar, and J. Orcutt. ROADNet: A network of SensorNets. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 579–587, 2006.

[46] J. Kay and P. Lauder. A fair share scheduler. *Communications of the ACM*, 31(1):44–55, 1988.

[47] J. Lindemann and G. Sandberg. An extendable GRID application portal. In *European Grid Conference (EGC)*. Springer Verlag, 2005.

[48] Maui Cluster Scheduler. http://www.clusterresources.com/products/maui/, September 2009.

[49] U. Maurer. Modelling a public-key infrastructure. *Lecture Notes in Computer Science*, 1146:325–350, 1996.

[50] F. Neubauer, A. Hoheisel, and J. Geiler. Workflow-based Grid applications. *Future Generation Computer Systems. The International Journal of Grid Computing: Theory, Methods and Applications*, 22(1-2):6–15, 2006.

[51] OASIS Open. Reference Model for Service Oriented Architecture 1.0. http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf, September 2009.

[52] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M.R. Pocock, A. Wipat, and P. Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[53] H. Overdick. The resource-oriented architecture. In *2007 IEEE Congress on Services (Services 2007)*, pages 340–347, 2007.

[54] I. Raicu, I.T. Foster, and Y. Zhao. Many-task computing for grids and supercomputers. In *Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS) 2008.*, pages 1–11, 2008.

[55] M. Russell, G. Allen, G. Daues, I. Foster, E. Seidel, J. Novotny, J. Shalf, and G. von Laszewski. The astrophysics simulation collaboratory: A science portal enabling community software development. *Cluster Computing*, 5(3):297–304, 2002.

[56] B. Segal, L. Robertson, F. Gagliardi, and F. Carminati. Grid computing: The European Data Grid Project. In *Nuclear Science Symposium Conference Record, 2000 IEEE*, volume 1, page 2/1, 2000.

[57] M. Snir, S.W. Otto, D.W. Walker, J. Dongarra, and S. Huss-Lederman. *MPI: The complete reference*. MIT Press Cambridge, MA, USA, 1995.

[58] H. Stockinger. Defining the grid: a snapshot on the current view. *The Journal of Supercomputing*, 42(1):3–17, 2007.

[59] A. Streit, D. Erwin, Th. Lippert, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, and Ph. Wieder. UNICORE - from project results to production grids. In L. Grandinetti, editor, *Grid Computing: The New Frontiers of High Performance Processing, Advances in Parallel Computing 14*, pages 357–376. Elsevier, 2005.

[60] T.A. Tatusova and T.L. Madden. BLAST 2 Sequences, a new tool for comparing protein and nucleotide sequences. *FEMS microbiology letters*, 174(2):247–250, 1999.

[61] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The Condor experience. *Concurrency Computat. Pract. Exper.*, 17(2–4):323–356, 2005.

[62] The Globus Project. An "ecosystem" of Grid components. http://www.globus.org/grid_software/ecology.php, September 2009.

[63] The Grid Infrastructure Research & Development (GIRD) project. Umeå University, Sweden. http://www.gird.se, September 2009.

[64] J. Treadwell. Open grid services architecture glossary of terms. In *Global Grid Forum, Lemont, Illinois, USA, GFD-I*, volume 44, pages 2–2, 2005.

[65] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall. A note on distributed computing. *Lecture Notes in Computer Science*, pages 49–64, 1997.

[66] R. Williams. Grids and the virtual observatory. *Grid Computing: Making the Global Infrastructure a Reality*, pages 837–858, 2003.