Designing Service-Based Resource Management Tools for a Healthy Grid Ecosystem*

Erik Elmroth, Francisco Hernández, Johan Tordsson, and Per-Olov Östberg

Dept. of Computing Science and HPC2N
Umeå University, SE-901 87 Umeå, Sweden
{elmroth, hernandf, tordsson, p-o}@cs.umu.se

Abstract. We present an approach for development of Grid resource management tools, where we put into practice internationally established high-level views of future Grid architectures. The approach addresses fundamental Grid challenges and strives towards a future vision of the Grid where capabilities are made available as independent and dynamically assembled utilities, enabling run-time changes in the structure, behavior, and location of software. The presentation is made in terms of design heuristics, design patterns, and quality attributes, and is centered around the key concepts of co-existence, composability, adoptability, adaptability, changeability, and interoperability. The practical realization of the approach is illustrated by five case studies (recently developed Grid tools) high-lighting the most distinct aspects of these key concepts for each tool. The approach contributes to a healthy Grid ecosystem that promotes a natural selection of "surviving" components through competition, innovation, evolution, and diversity. In conclusion, this environment facilitates the use and composition of components on a per-component basis.

1 Introduction

In recent years, the vision of the Grid as the general-purpose, service-oriented infrastructure for provisioning of computing, data, and information capabilities has started to materialize in the convergence of Grid and Web services technologies. Ultimately, we envision a Grid with open and standardized interfaces and protocols, where independent Grids can interoperate, virtual organizations co-exist, and capabilities be made available as independent utilities.

However, there is still a fundamental gap between the technology used in major production Grids and recent technology developed by the Grid research community. While current research directions focus on user-centric and service-oriented infrastructure design for scenarios with millions of self-organizing nodes, current production Grids are often more monolithic systems with stronger inter-component dependencies.

* 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.

We present an approach to Grid infrastructure component development, where internationally established high-level views of future Grid architectures are put into practice. Our approach addresses the future vision of the Grid, while enabling easy integration into current production Grids. We illustrate the feasibility of our approach by presenting five case studies.

The outline of the rest of the paper is as follows. Section 2 gives further background information, including our vision of the Grid, a characterization of competitive factors for Grid software, and a brief review of internationally established conceptual views of future Grid architectures. Section 3 presents our approach to Grid infrastructure development, which complies with these views. The realization of this approach for specific components is illustrated in Section 4, with a brief presentation of five tools recently developed within the Grid Infrastructure Research & Development (GIRD) project [26]. These are Grid tools or toolkits for resource brokering [9–11], job management [7], workflow execution [8], accounting [16, 24], and Grid-wide fairshare scheduling [6].

2 Background and Motivation

Our approach to Grid infrastructure development is driven by the need and opportunity for a general-purpose infrastructure. This infrastructure should facilitate flexible and transparent access to distributed resources, dynamic composition of applications, management of complex processes and workflows, and operation across geographical and organizational boundaries. Our vision is that of a large evolving system, realized as a Service-Oriented Architecture (SOA) that enables provisioning of computing, data, and information capabilities as utility-like services serving business, academia, and individuals. From this point of departure, we elaborate on fundamental challenges that need to be addressed to realize this vision.

2.1 Facts of life in Grid environments

The operational context of a Grid environment is harsh, with heterogeneity in resource hardware, software, ownerships, and policies. The Grid is distributed and decentralized by nature, and any single point of control is impossible not only for scalability reasons but also since resources are owned by different organizations. Furthermore, as resource availability varies, resources may at any time join or leave the Grid. Information about the set of currently available resources and their status will always to some extent be incomplete or outdated.

Actors have different incentives to join the Grid, resulting in asymmetric resource sharing relationships. Trust is also asymmetric, which in scenarios with cross trust-domain orchestration of multiple resources that interact beyond the client-server model, gives rise to complex security challenges.

Demand for resources typically exceed supply, with contention for resources between users as a consequence. The Grid user community at large is disparate

in requirements and knowledge, necessitating the development of wide ranges of user interfaces and access mechanisms. All these complicating factors add up to an environment where errors are rule rather than exception.

2.2 A General-purpose Grid ecosystem

Recently, a number of organizations have expressed views on how to realize a single and fully open architecture for the future Grid. To a large extent, these expressions conform to a single view of a highly dynamic service-oriented infrastructure for general-purpose use.

One such view proposes the model of a healthy ecosystem of Grid components [25], where components occupy niches in the ecosystem and are designed for component-by-component selection by developers, administrators, and end-users. Components are developed by the Grid community at large and offer sensible functionality, available for easy integration in high-level tools or other software. In the long run, competition, innovation, evolution, and diversity lead to natural selection of "surviving" components, whereas other components eventually fade out or evolve into different niches.

European organizations, such as the Next Generation Grids expert group [12] and NESSI [23], have focused on a common architectural view for Grid infrastructure, possibly with a more emphasized business focus compared to previous efforts. Among their recommendations is a strong focus on SOAs where services can be dynamically assembled, thus enabling run-time changes in the structure, behavior, and location of software. The view of services as utilities includes directly and immediately usable services with established functionality, performance, and dependability. This vision goes beyond that of a prescribed layered architecture by proposing a multi-dimensional mesh of concepts, applying the same mechanisms along each dimension across the traditional layers.

In common for these views are, for example, a focus on composable components rather than monolithic Grid-wide systems, as well as a general-purpose infrastructure rather than application- or community-specific systems. Examples of usage range from business and academic applications to individual's use of the Grid. These visions also address some common issues in current production Grid infrastructures, such as interoperability and portability problems between different Grids, as well as limited software reuse. Before detailing our approach to Grid software design, which complies with the views presented above, we elaborate on key factors for software success in the Grid ecosystem.

2.3 Competitive factors for software in the Grid ecosystem

In addition to component-specific functional requirements, which obviously differ for different types of components, we identify a set of general quality attributes (also known as non-functional requirements) that successful software components should comply with. The success metrics considered here are the amount of users and the sustainability of software.

In order to attract the largest possible user community, usability aspects such as availability, ease of installation, understandability, and quality of documentation and support are important. With the dynamic and changing nature of Grid environments, flexibility and the ability to adapt and evolve is vital for the survival of a software component. Competitive factors for survival include changeability, adaptability, portability, interoperability, and integrability. These factors, along with mechanisms used to improve software quality with respect to them, are further discussed in Section 3. Other criteria, relating to sustainability, include the track record of both components and developers as well as the general reputation of the latter in the user community.

Quality attributes such as efficiency (with emphasis on scalability), reliability, and security also affect the software success rate in the Grid ecosystem. These attributes are however not further discussed herein.

3 Grid Ecosystem Software Development

In this section we present our approach to building software well-adjusted to the Grid ecosystem. The presentation is structured into five groups of software design heuristics, design patterns, and quality attributes that are central to our approach. All definitions are adapted to the Grid ecosystem environment, but are derived from, and conform to, the ISO/IEC 9126-1 standard [20].

3.1 Co-existence – Grid ecosystem awareness

Co-existence is defined as the ability of software to co-exist with other independent softwares in a shared resource environment. The behavior of a component well adjusted to the Grid ecosystem is characterized by non-intrusiveness, respect for niche boundaries, replaceability, and avoidance of resource overconsumption.

When developing new Grid components, we identify the purpose and boundaries of the corresponding niches in order to ensure the components' place and role in the ecosystem. By stressing non-intrusiveness in the design, we strive to ensure that new components do not alter, hinder, or in any other way affect the function of other components in the system. While the introduction of new software into an established ecosystem may, through fair competition, reshape, create, or eliminate niches, it is still important for the software to be able to cooperate and interact with neighboring components.

By the principle of decentralization, it is crucial to avoid making assumptions of omniscient nature and not to rely on global information or control in the Grid. By designing components for a user-centric view of systems, resources, component capabilities, and interfaces, we emphasize decentralization and facilitate component co-existence and usability.

3.2 Composability – software reuse in the Grid ecosystem

Composability is defined as the capability of software to be used both as individual components and as building blocks in other systems. As systems may

themselves be part of larger systems, or make use of other systems' components, composability becomes a measure of usefulness at different levels of system design. Below, we present some design heuristics that we make use of in order to improve software composability.

By designing components and component interactions in terms of interfaces rather than functionality, we promote the creation of components with well-defined responsibilities and provision for module encapsulation and interface abstraction. We strive to develop simple, single-purpose components achieving a distinct separation of concerns and a clear view of service architectures. Implementation of such components is faster and less error-prone than more complex designs. Autonomous components with minimized external dependencies make composed systems more fault tolerant as their distributed failure models become simpler.

Key to designing composable software is to provision for software reuse rather than reinvention. Our approach, leading to generic and composable tools well adjusted to the Grid ecosystem, encourages a model of software reuse where users of components take what they need and leave the rest. Being decentralized and distributed by nature, SOAs have several properties that facilitate the development of composable software.

3.3 Adoptability – Grid ecosystem component usability

Adoptability is a broad concept enveloping aspects such as end-user usability, ease of integration, ease of installation and administration, level of portability, and software maintainability. These are key factors for determining deployment rate and niche impact of a software.

As high software usability can both reduce end-user training time and increase productivity, it has significant impact on the adoptability of software. We strive for ease of system installation, administration, and integration (e.g., with other tools or Grid middlewares), and hence reduce the overhead imposed by using the software as stand-alone components, end-user tools, or building blocks in other systems. Key adoptability factors include quality of documentation and client APIs, as well as the degree of openness, complexity, transparency and intrusiveness of the system.

Moreover, high portability and ease of migration can be deciding factors for system adoptability.

3.4 Adaptability and Changeability – surviving evolution

Adaptability, the ability to adapt to new or different environments, can be a key factor for improving system sustainability. *Changeability*, the ability for software to be changed to provide modified behavior and meet new requirements, greatly affects system adaptability.

By providing mechanisms to modify component behavior via configuration modules, we strive to simplify component integration and provide flexibility in,

and ease of, customization and deployment. Furthermore, we find that the use of policy plug-in modules which can be provided and dynamically updated by third parties are efficient for making systems adaptable to changes in operational contexts. By separating policy from mechanism, we facilitate for developers to use system components in other ways than originally anticipated and software reuse can thus be increased.

3.5 Interoperability – interaction within the Grid ecosystem

Interoperability is the ability of software to interact with other systems. Our approach includes three different techniques for making our components available, making them able to access other Grid resources, and making other resources able to access our components, respectively. Integration of our components typically only requires the use of one or two of these techniques.

Whenever feasible, we leverage established and emerging Web and Grid services standards for interfaces, data formats, and architectures. Generally, we formulate integration points as interfaces expressing required functionality rather than reflecting internal component architecture. Our components are normally made available as Grid services, following these general principles.

For our components to access resources running different middlewares, we combine the use of customization points and design patterns such as Adapter and Chain of Responsibility [15]. Whenever possible, we strive to embed the customization points in our components, simplifying component integration with one or more middlewares.

In order to make existing Grid softwares able to access our components, we strive to make external integration points as few, small, and well-defined as possible, as these modifications need to be applied to external softwares.

4 Case Studies

We illustrate our approach to software development by brief presentations of five tools or toolkits recently developed in the GIRD project [26]. The presentations describe the overall tool functionality and high-light the most significant characteristics related to the topics discussed in Section 3.

All tools are built to operate in a decentralized Grid environment with no single point of control. They are furthermore designed to be non-intrusive and can coexist with alternative mechanisms. To enhance adoptability of the tools, user guides, administrator manuals, developer APIs, and component source code are made available online [26]. As these adoptability measures are common for all projects, the adoptability characteristics are left out of the individual project presentations.

The use of SOAs and Web services naturally fulfills many of the composability requirements outlined in Section 3. The Web service toolkit used is the Globus Toolkit 4 (GT4) Java WS Core, which provides an implementation of the Web Services Resource Framework (WSRF). Notably, the fact that our tools are made

available as GT4-based Web services should not be interpreted as been built primarily for use in GT4-based Grids. On the contrary, their design is focused on generality and ease of middleware integration.

4.1 Job Submission Service (JSS)

The JSS is a feature-rich, standards-based service for cross-middleware job submission, providing support, e.g., for advance reservations and co-allocation. The service implements a decentralized brokering policy, striving to optimize the job performance for individual users by minimizing the response time for each submitted job. In order to do this, the broker makes an a priori estimation of the whole, or parts of, the Total Time to Delivery (TTD) for all resources of interest before making the resource selection [9–11].

Co-existence: The non-intrusive decentralized resource broker handles each job isolated from the jobs of other users. It can provide quality of service to end-users despite the existence of competing job submission tools.

Composability: The JSS is composed of several modules, each performing a well-defined task in the job submission process, e.g., resource discovery, reservation negotiation, resource selection, and data transfer.

Changeability and adaptability: Users of the JSS can specify additional information in job request messages to customize and fine-tune the resource selection process. Developers can replace the resource brokering algorithms with alternative implementations.

Interoperability: The architecture of the JSS is based on (emerging) standards such as JSDL, WSRF, WS-Agreement, and GLUE. It also includes customization points, enabling the use of non-standard job description formats, Grid information systems, and job submission mechanisms. The latter two can be interfaced despite differences in data formats and protocols. By these mechanisms, the JSS can transparently submit jobs to and from GT4, NorduGrid/ARC, and LCG/gLite.

4.2 Grid Job Management Framework (GJMF)

The GJMF [7] is a framework for efficient and reliable processing of Grid jobs. It offers transparent submission, control, and management of jobs and groups of jobs on different middlewares.

Co-existence: The user-centric GJMF design provides a view of exclusive access to each service and enforces a user-level isolation which prohibits access to other users' information. All services in the framework assume shared access to Grid resources. The resource brokering is performed without use of global information, and includes back-off behaviors for Grid congestion control on all levels of job submission.

Composability: Orchestration of services with coherent interfaces provides transparent access to all capabilities offered by the framework. The functionality for job group management, job management, brokering, Grid information system access, job control, and log access are separated into autonomous services.

Changeability and adaptability: Configurable policy plug-ins in multiple locations allow customization of congestion control, failure handling, progress monitoring, service interaction, and job (group) prioritizing mechanisms. Dynamic service orchestration and fault tolerance is provided by each service being capable of using multiple service instances. For example, the job management service is capable of using several services for brokering and job submission, automatically switching to alternatives upon failures.

Interoperability: The use of standardized interfaces such as JSDL as job description format, OGSA BES for job execution, and OGSA RSS for resource selection improves interoperability and replaceability.

4.3 Grid Workflow Execution Engine (GWEE)

The GWEE [8] is a light-weight and generic workflow execution engine that facilitates the development of application-oriented end-user workflow tools. The engine is light-weight in that it focuses only on workflow execution and the corresponding state management. This project builds on experiences gained while developing the Grid Automation and Generative Environment (GAUGE) [19, 17].

Co-existence: The engine operates in the narrow niche of workflow execution. Instead of attempting to replace other workflow tools, the GWEE provides a means for accessing advanced capabilities offered by multiple Grid middlewares. The engine can process multiple workflows concurrently without them interfering with each other. Furthermore, the engine can be shared among multiple users, but only the creator of a workflow instance can monitor and control that workflow.

Composability: The main responsibilities of the engine, managing task dependencies, processing tasks on Grid resources, and managing workflow state, are performed by separate modules.

Adaptability and Changeability: Workflow clients can monitor executing workflows both by synchronous status requests and by asynchronous notifications. Different granularities of notifications are provided to support specific client requirements – from a single message upon workflow completion to detailed updates for each task state change.

Interoperability: The GWEE is made highly interoperable with different middlewares and workflow clients through the use of two types of plug-ins. Currently, it provides middleware plug-ins for execution of computational tasks in GT4 and in the GJMF, as well as GridFTP file transfers. It also provides plug-ins for transforming workflow languages into its native language, as currently has been done for the Karajan language. The Chain of Responsibility design pattern allows concurrent usage of multiple implementations of a particular plug-in.

4.4 SweGrid Accounting System (SGAS)

SGAS allocates Grid capacity between user groups by coordinated enforcement of Grid-wide usage limits [24, 16]. It employs a credit-based allocation model

where Grid capacity is granted to projects via Grid-wide quota allowances. The Grid resources collectively enforce these allowances in a soft, real-time manner. The main SGAS components are a Bank, a logging service (LUTS), and a quota-aware authorization tool (JARM), the latter to be integrated on each Grid resource.

Co-existence: SGAS is built as stand-alone Grid services with minimal dependencies on other software. Normal usage is not only non-intrusive to other software but also to usage policies, as resource owners retain ultimate control over local resource policies, such as strictness of quota enforcement.

Composability: There is a distinct separation of concerns between the Bank and the LUTS, for managing usage quotas and logging usage data, respectively. They can each be used independently.

Changeability and adaptability: The Bank can be used to account for any type of resource consumption and with any price-setting mechanism, as it is independent of the mapping to the abstract "Grid credit" unit used. The Bank can also be changed from managing pre-allocations to accumulating costs for later billing. The JARM provides customization points for calculating usage costs based on different pricing models. The tuning of the quota enforcement strictness is facilitated by a dedicated customization point.

Interoperability: The JARM has plug-in points for middleware-specific adapter code, facilitating integration with different middleware platforms, scheduling systems, and data formats. The middleware integration is done via a SOAP message interceptor in GT4 GRAM and via an authorization plug-in script in the NorduGrid/ARC GridManager. The LUTS data is stored in the OGF Usage Record format.

4.5 Grid-Wide Fairshare Scheduling System (FSGrid)

FSGrid is a Grid-wide fairshare scheduling system that provides three-party QoS support (user, resource-owner, VO-authority) for enforcement of locally and globally scoped share policies [6]. The system allows local resource capacity as well as global Grid capacity to be logically divided among different groups of users. The policy model is hierarchical and sub-policy definition can be delegated so that, e.g., a VO can partition its share among its projects, which in turn can divide their shares among users.

Co-existence: The main objective of FSGrid is to facilitate for distributed resources to collaboratively schedule jobs for Grid-wide fairness. FSGrid is non-intrusive in the sense that resource owners retain ultimate control of how to perform the scheduling on their local resources.

Composability: FSGrid includes two stand-alone components with clearly separated concerns for maintaining a policy tree and to log usage data, respectively. In fact, the logging component in current use is the LUTS originally developed for SGAS, illustrating the potential for reuse of that component.

Changeability and adaptability: A customizable policy engine is used to calculate priority factors based on a runtime policy tree with information about

resource pre-allocations and previous usage. The priority calculation can be customized, e.g., in terms of length, granularity, and rate of aging of usage history. The administration of the policy tree is flexible as sub-policy definition can be delegated to, e.g., VOs and projects.

Interoperability: Besides the integration of the LUTS (see Section 4.4), FSGrid includes a single external point of integration, as a fair-share priority factor call-out to FSGrid has to be integrated in the local scheduler on each resource.

5 Related Work

Despite the large amount of Grid related projects to date, just a few of these have shared their experiences regarding software design and development approaches. Some of these projects have focused on software architecture. In a survey by Filkenstein et al. [13], existing data-Grids are compared in terms of their architectures, functional requirements, and quality attributes. Cakic et al. [2] describe a Grid architectural style and a light-weight methodology for constructing Grids. Their work is based on a set of general functional requirements and quality attributes that derives an architectural style that includes information, control, and execution. Mattmann et al. [22] analyze software engineering challenges for large-scale scientific applications, and propose a general reference architecture that can be instantiated and adapted for specific application domains. We agree on the benefits obtained with a general architecture for Grid components to be instantiated for specific projects, however, our focus is on the inner workings of the components making up the architecture.

The idea of software that evolves due to unforeseen changes in the environment also appears in the literature. In the work by Smith et al. [3], the way software is modified over time is compared with Darwinian evolution. In this work, the authors discuss the best-of-breed approach, where an organization collects and assembles the most suitable software component from each niche. The authors also construct a taxonomy of the "species" of enterprise software. A main difference between this work and our contribution is that our work focuses on software design criteria.

Other high-level visions of Grid computing include that of interacting autonomous software agents [14]. One of the characteristics of this vision is that software engineering techniques employed for software agents can be reused with little or no effort if the agents encompasses the service's vision [21]. A different view on agent-based software development for the Grid is that of evolution based on competition between resource brokering agents [4]. These projects differ from our contribution as our tools have a stricter focus on functionality (being well-adjusted to their respective niches).

Finally, it is also important to notice that there are a number of tools that simplify the development of Grid software. These tools facilitate, for example, implementation [18], unit testing [5], and automatic integration [1].

6 Concluding Remarks

We explore the concept of the Grid ecosystem, with well-defined niches of functionality and natural selection (based on competition, innovation, evolution, and diversity) of software components within the respective niches. The Grid ecosystem facilitates the use and composition of components on a per-component basis. We discuss fundamental requirements for software to be well-adjusted to this environment and propose an approach to software development that complies with these requirements. The feasibility of our approach is demonstrated by five case studies. Future directions for this work include further exploration of processes and practices for development of Grid software.

7 Acknowledgements

We acknowledge Magnus Eriksson for valuable feedback on software engineering standardization matters.

References

1. M-E. Bégin, G. Diez-Andino, A. Di Meglio, E. Ferro, E. Ronchieri, M. Selmi, and M. Zurek. Build, configuration, integration and testing tools for large software projects: ETICS. In N. Guelfi and D. Buchs, editors, *Rapid Integration of Software Engineering Techniques,* LNCS 4401, pp. 81–97. Springer-Verlag, 2007.
2. J. Cakic and R. F. Paige. Origins of the Grid architectural style. In *Engineering of Complex Computer Systems. 11th IEEE Int. Conference, IECCS 2006*, pp. 227–235. IEEE CS Press, 2006.
3. J. Smith David, W. E. McCarthy, and B. S. Sommer. Agility – the key to survival of the fittest in the software market. *Commun. ACM*, 46(5):65–69, 2003.
4. C. Dimou and P. A. Mitkas. An agent-based metacomputing ecosystem. http://issel.ee.auth.gr/ktree/Documents/Root Folder/ISSEL/Publications/Biogrid An Agent-based Metacomputing Ecosystem.pdf, visited October 2007.
5. A. Duarte, W. Cirne, F. Brasileiro, and P. Machado. GridUnit: software testing on the Grid. In K.M. Anderson, editor, *Software Engineering. 28th Int. Conference, ICSE 2006*, pp. 779–782. ACM Press, 2006.
6. E. Elmroth and P. Gardfjäll. Design and evaluation of a decentralized system for Grid-wide fairshare scheduling. In H. Stockinger et al., editors, *First International Conference on e-Science and Grid Computing*, pp. 221–229. IEEE CS Press, 2005.
7. 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*, pp. 175–184. Springer-Verlag, 2007.
8. 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. 7th Int. Conference, PPAM 2007*. Lecture notes in Computer Science, Springer Verlag, 2007 (to appear).
9. E. Elmroth and J. Tordsson. An interoperable, standards-based Grid resource broker and job submission service. In H. Stockinger et al., editors, *First International Conference on e-Science and Grid Computing*, pp. 212–220. IEEE CS Press, 2005.

10. E. Elmroth and J. Tordsson. A standards-based Grid resource brokering service supporting advance reservations, coallocation and cross-Grid interoperability. *Submitted to Concurrency and Computation: Practice and Experience*, 2006.
11. E. Elmroth and J. Tordsson. A 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*, 2008, to appear.
12. Expert Group on Next Generation Grids 3 (NGG3). Future for European Grids: Grids and service oriented knowledge utilities. Vision and research directions 2010 and beyond, 2006. ftp://ftp.cordis.lu/pub/ist/docs/grids/ngg3_eg_final.pdf, visited October 2007.
13. A. Finkelstein, C. Gryce, and J. Lewis-Bowen. Relating requirements and architectures: a study of data-grids. *J. Grid Computing*, 2(3):207–222, 2004.
14. I. Foster, N. R. Jennings, and C. Kesselman. Brain meets brawn: why Grid and agents need each other. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, pp. 8–15. IEEE CS Press, 2004.
15. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
16. P. Gardfjäll, E. Elmroth, L. Johnsson, O. Mulmo, and T. Sandholm. Scalable Grid-wide capacity allocation with the SweGrid Accounting System (SGAS). *Concurrency and Computation: Practice and Experience*, (accepted) 2007.
17. Z. Guan, F. Hernández, P. Bangalore, J. Gray, A. Skjellum, V. Velusamy, and Y. Liu. Grid-Flow: a Grid-enabled scientific workflow system with a petri-net-based interface. *Concurrency Computat.: Pract. Exper.*, 18(10):1115–1140, 2006.
18. S. Hastings, S. Oster, S. Langella, D. Ervin, T. Kurc, and J. Saltz. Introduce: an open source toolkit for rapid development of strongly typed Grid services. *J. Grid Computing*, 5(4):407–427, 2007.
19. F. Hernández, P. Bangalore, J. Gray, Z. Guan, and K. Reilly. GAUGE: Grid Automation and Generative Environment. *Concurrency Computat.: Pract. Exper.*, 18(10):1293–1316, 2006.
20. ISO/IEC. Software engineering - Product quality - Part 1: Quality model. International standard ISO/IEC 9126-1. 2001.
21. P. Leong, C. Miao, and B-S. Lee. Agent oriented software engineering for Grid computing. In *Cluster Computing and the Grid. 6th IEEE Int. Symposium, CCGRID 2006*. IEEE CS Press, 2006.
22. C. A. Mattmann, D. J. Crichton, N. Medvidovic, and S. Hughes. A software architecture-based framework for highly distributed and data intensive scientific applications. In K.M. Anderson, editor, *Software Engineering. 28th Int. Conference, ICSE 2006*, pp. 721–730. ACM Press, 2006.
23. Networked European Software and Services Initiative (NESSI). http://www.nessi-europe.com, visited October 2007.
24. T. Sandholm, P. Gardfjäll, E. Elmroth, L. Johnsson, and O.Mulmo. A service-oriented approach to enforce Grid resource allocations. *International Journal of Cooperative Information Systems*, 15(3):439–459, 2006.
25. The Globus Project. An "ecosystem" of Grid components. http://www.globus.org/grid_software/ecology.php, visited October 2007.
26. The Grid Infrastructure Research & Development (GIRD) project. Umeå University, Sweden. http://www.gird.se, visited October 2007.