XSEDE Architecture

Level 1 and 2 Decomposition

Prepared by:

Felix Bachmann

Ian Foster

Andrew Grimshaw

Dave Lifka

Morris Riedel

Steve Tuecke

February 21, 2012

Version 1

Abstract

This document presents the Levels 1 and 2 decompositions of Version 1 of the XSEDE architecture, as defined by the XSEDE architecture team based on requirements obtained by broad stakeholder consultation. The Level 1 decomposition identifies the nature and purpose of the three principal layers of the architecture: the resources layer, which encompasses the diverse physical and virtual resources provided by XSEDE service providers; the services layer, which provides access to those resources via well-defined network protocols; and the access layer, which facilitates the use of resources via a range of end-user-oriented and programmer-oriented interfaces. The Level 2 decomposition identifies the capabilities to be provided within each layer. At the access layer, these capabilities encompass thin client graphical user interfaces, thick client graphical user interfaces, command line interfaces, application programmer interfaces, and file system mechanisms. Services layer capabilities encompass execution management, discovery and information, identity, accounting and allocation, data management, infrastructure services, and help desk and ticketing. Finally, the resources layer comprises computers, file systems, and the like. More detailed specifications of the capabilities to be provided in the services and access layers will be provided in a separate XSEDE Architecture Level 3 Decomposition document. The architecture will evolve over time to meet new requirements, at which time new versions of this and other related documents will be produced.

A Introduction

The XSEDE architecture team is charged with translating requirements and use cases obtained from XSEDE stakeholders into architectural definitions. These definitions are provided at three levels of details: high-level concepts (the Level 1 decomposition); listings of classes of capabilities (Level 2); and detailed specifications of the interfaces, capabilities, and quality attributes of specific components (Level 3 [2]). The architecture team is also charged with managing the process by which changes in requirements over time are translated into changes in architecture.

The XSEDE architecture definition process is highly consultative. It is driven first of all by requirements gathered from a wide range of stakeholders. Where requirements seem unclear, the architecture team also engages with relevant groups to elicit detailed use cases. Requirements and use cases together provide one set of constraints under which the architectural definition process operates. Other constraints derive from the technological state of the art (there is no point defining an architecture that depends on technologies that do not exist) and from the goal of ensuring that architectural change, when required, does not unnecessarily inconvenience XSEDE users. Given these constraints, the XSEDE architecture team then works to define a workable architecture, ensuring that requirements are met—or, if they cannot be met, that this fact is known; and that consistency is achieved across different architectural elements.

This Version 1 of the XSEDE Architecture Level 1 and 2 Decompositions is based on Version 1 of the XSEDE user requirements, as documented [9]. Those requirements will be updated on a yearly basis, or more frequently if required, at which time the architecture team will determine whether changes to the XSEDE architecture definition are required. Architectural evolution is highly likely to change Level 3 capabilities, less likely to change Level 2 components, and unlikely to alter the Level 1 structure.

B Level 1 Decomposition

The Level 1 decomposition of the XSEDE architecture identifies three distinct layers: resource, services layer, and access, as illustrated in Figure 1.

Starting at the bottom, the **resources layer** comprises the physical and virtual resources to which we wish to enable convenient remote access: computers, file systems, and the like. These components are frequently highly heterogeneous in their architectures, configurations and policies, and can require different, even site-specifics, mechanisms for their use. An important goal of the XSEDE architecture is to encapsulate local heterogeneities in a way that enables XSEDE users to access resources without detailed local knowledge.

Next, the **services layer** is, quite literally, the core of the architecture. This layer defines an important class of encapsulation mechanisms, namely the well-defined protocol specifications that define the interfaces to the various XSEDE services. For example, an execution management service defines the protocol to be employed to request the execution of a job on an XSEDE resource. Services then use resource-specific mechanisms to interact with individual resources at the **resources layer**: for example, by using scheduler-specific functions to enqueue and then monitor and control jobs.

Finally, the **access layer** provides user-oriented interfaces to services, ranging from thick and thin client graphical user interfaces to libraries that implement application programmer interfaces (APIs), programs that implement command line interfaces (CLIs), and file system mechanisms. Access layer components may simplify interactions with service layer capabilities and/or provide more sophisticated functionality that builds on top of those capabilities.

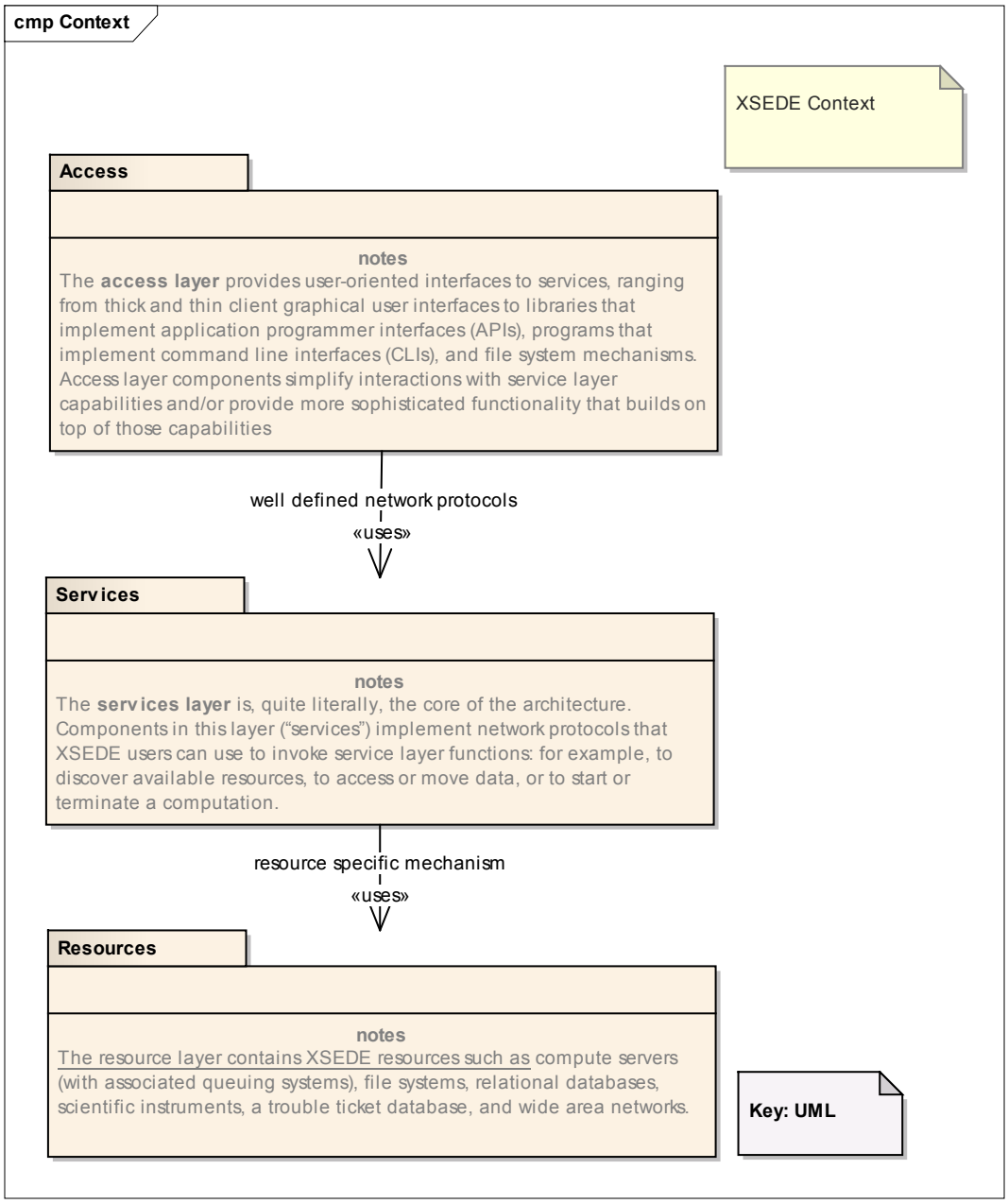

Figure 1. The three layers of the XSEDE architecture: access, services, and resources.

This layered organization allows the XSEDE architecture to hide heterogeneity. If an XSEDE user employs service-layer protocols (either directly, or via access layer components) to move data, submit jobs, etc., then those same mechanisms will continue to work even as the underlying resources are upgraded or new resources are added, and/or as a user moves from one resource to another.

Illustrative Example #1: A user invokes a workflow tool to execute a parameter study on XSEDE resources. The workflow tool uses an **access layer** API to invoke a **services layer** job execution and file staging service, which itself manages the submission and execution of user jobs on one or more **resources layer** computer systems (e.g., supercomputers at XSEDE sites).

Illustrative Example #2: A user invokes an **access layer** command line tool to submit a job to a meta-scheduler at the **services layer**. The meta-scheduler service then uses an **access layer** API to invoke a

services layer information service to determine load and resource use policy and then invokes **services layer** job execution and file staging services, which themselves manage the submission and execution of user jobs on one or more **resources layer** computer systems (e.g., supercomputers at XSEDE sites).

Note regarding direct resource access: Some XSEDE users, particularly power users, use *ssh* to login directly to an XSEDE supercomputer, and then local commands (e.g., *qsub*) to run jobs on the supercomputer. This mode of use is not prohibited, and indeed we expect it to continue to be supported in the future. However, those mechanisms are not part of the value added by the XSEDE architecture and thus we do not describe them here.

C Level 2 Decomposition

The Level 2 decomposition further refines the descriptions in Section B, identifying the broad classes of capabilities required at the Access Layer and Services Layer to meet user requirements.

C.1 Access Layer

As discussed above, an access layer component is a program or library that uses protocols implemented by service layer components to interact with one or more of those components—as, for example, when an access layer file transfer Graphical User Interface (GUI) tool generates messages to a file transfer service to request that a data movement operation be performed.

XSEDE will provide a set of supported access layer components that are designed to meet the needs of typical XSEDE users. The XSEDE Architecture Level Three Decomposition [2] details these components and their capabilities. However, the definition of XSEDE-supported access layer components does not preclude others from developing and using other such components. (Indeed, an important feature of the XSEDE architecture is that such innovation is enabled and encouraged.) For example, high energy physicists working on the ATLAS experiment might adapt the ATLAS Distributed Computing System [7] to use XSEDE service-level protocols to access XSEDE resources.

Access layer components fall into five broad categories (Figure 2): thin client GUIs, thick client GUIs, command line interfaces (CLIs), application programming interfaces (APIs), and file system mechanisms.

Thin client GUIs are access layer components that are accessed via a web browser and thus do not require users to install any software beyond a standard Web browser on their workstation/desktop/PDA in order to access XSEDE resources. Examples of services that provide thin client GUIs include the XSEDE User Portal, Globus Online [1, 4], and many gateways.

Thick client GUIs are an alternative to thin clients. These GUIs require that some application beyond a Web browser be installed and executed on the machine from which XSEDE services are to be accessed. For example, a thick client may be installed on an XSEDE Service Provider (SP) login node, on a departmental file server, on the user's desktop, or on the user's PDA. Examples include the Genesis II GUI and the UNICORE 6 Rich Client (URC) [3].

Command line interfaces (CLIs) are tools that that allow XSEDE resources and services to be accessed from the command line or via scripting languages such as BASH. Examples include the UNICORE Command Line Client (UCC) [8], the Globus Toolkit CLI (e.g., globus-url-copy), the Globus Online CLI, and the Genesis II grid shell [6]. CLIs are typically implemented by programs that must be installed on the user's computer. One exception is the Globus Online CLI, which is invoked remotely via SSH.

Application programming interfaces (APIs) provide language-specific interfaces for interacting with XSEDE services. APIs are implemented by libraries that can be linked with application programs. Examples include the Simple API for Grid Applications (SAGA) bindings for C, Python, and Java [5]; the Genesis II Java bindings; and, the jGlobus libraries for Java.

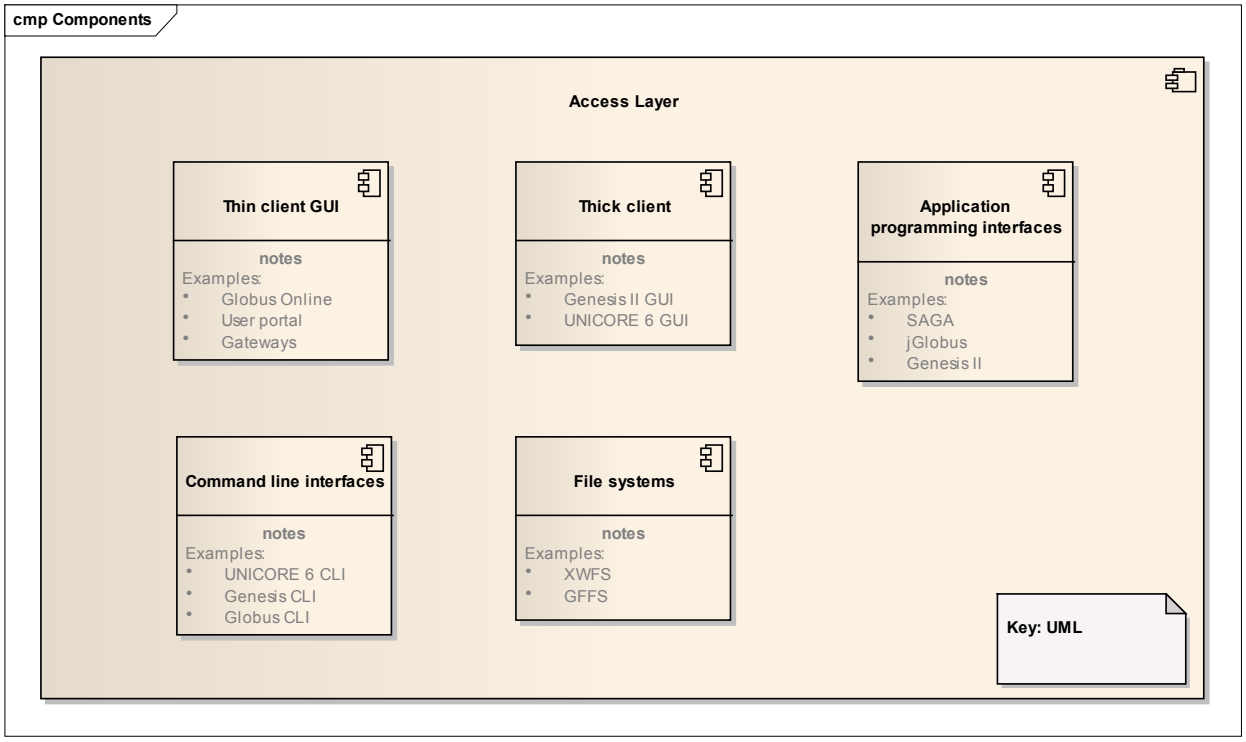

Figure 2. The access layer consists of a diverse set of components that facilitate access to XSEDE services. The items listed in "notes" are examples in each category.

File system mechanisms: The file system paradigm and interfaces, which are familiar to most users, clearly specified (POSIX - Portable Operating System Interface) and ubiquitous, can be leveraged to provide access to XSEDE services and resources. Examples (beyond local file systems) include the XSEDE Wide File System (XWFS) and the Global Federated File System (GFFS).

C.2 Services Layer

The services layer (Figure 3) implements the interfaces that both users and other services invoke, typically via access layer components. Some services are deployed at many or all XSEDE SPs (e.g., file access services), while others may be deployed in just one location (e.g., the XSEDE User Portal). The XSEDE Architecture Level Three Decomposition [2] details XSEDE services and their capabilities.

Execution Management Services (EMS) are concerned with instantiating and managing to completion work units that may consist of single activities, sets of independent activities, or workflows. EMS provides for execution of work units, including their placement, provisioning, and lifetime management. EMS tasks may include, but are not limited to, the following: finding execution candidate locations, selecting execution location, preparing for execution, initiating execution, and/or managing execution.

Discovery and Information Services address needs to find resources based on descriptive metadata and to subscribe to events or changes in resource status. Such services can include registries into which information (meta-data) about a resource can be published and that can subsequently be queried to find resources that match specified criteria, or can provide pub-sub style interactions with streams of message events. Descriptive metadata may range from static (changing slowly, such as the name of a machine) to dynamic (such as the current load on a machine). Similarly the information may be intended for either human consumption or for application/tool consumption. It also includes the notions of information aggregation and dis-aggregation.

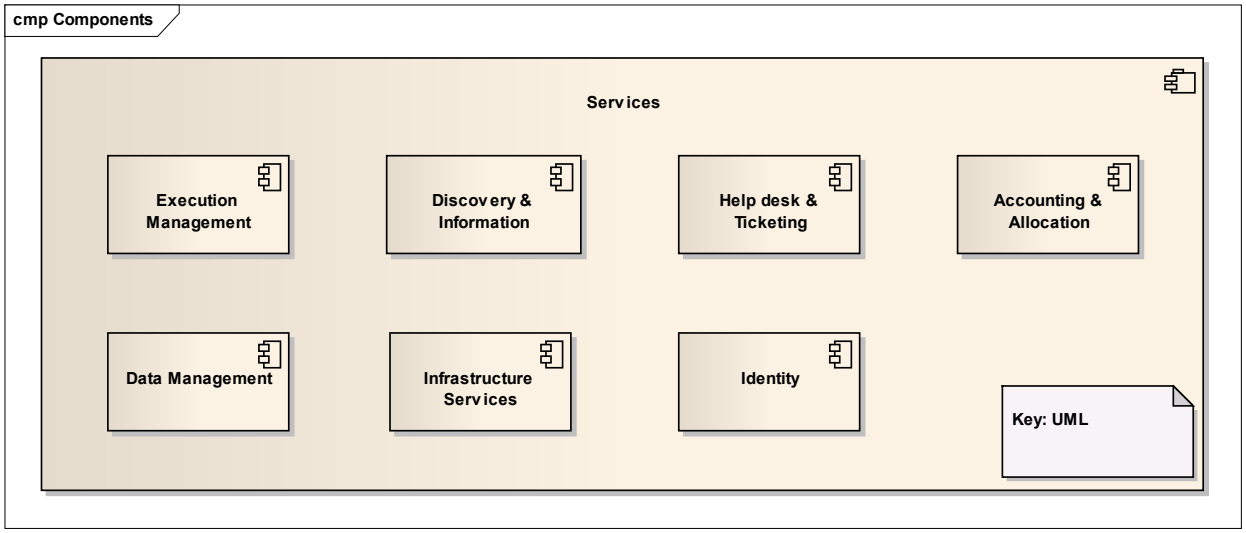

Figure 3. The services in the services layer are decomposed into seven broad categories: Execution Management; Discovery and Information; Identity; Accounting and Allocation; Data Management; Infrastructure Services and Mechanisms; and Help Desk and Ticketing.

Identity services provide for identifying and associating attributes with individuals, services, groups, roles, communities, and resources (e.g., a software component or set of computers). This information is often used for making authorization decisions, maintaining audit trails, logging, accounting, and searching for resources owned by, or accessible to, a particular individual, service, or group. Important aspects include the strength, ubiquity, interoperability, and performance of the identity mechanisms.

Accounting and Allocation is concerned with keeping track of resource consumption, and what consumption is allowed. These services provide XSEDE service providers and users with capabilities that are analogous to bank accounts and charge cards, such as currency (e.g., normalized resource units such as normalized CPU hours, gigabyte/months, and gigabytes transferred and limits). Operations include putting "money" into accounts, authorizing transfers, querying balances, and clearing transactions.

Data Management includes the access and manipulation of both user data and system data residing at service providers, including XSEDE resource providers, campuses, research labs, other infrastructures, and third party providers such as cloud storage providers. Functionality includes (subject to authorization) create, read, update, delete (CRUD), copy, replicate data, etc. Data types include, for example, user flat files, user relational data, system databases (log files, accounting records, event logs, tickets, streams, block oriented), data in archives, data on instruments, and streaming data from instruments and sensors. Data management also includes metadata management and search capabilities. Data management functions also include data and allocation reservations.

Infrastructure Services include naming and binding services, resource introspection and reflection services, and fault detection and recovery services.

Help Desk and Ticketing services provide interfaces for ticket management (e.g., submit, check status, change status, comment, query) and for help desk federation (e.g., ticket forwarding and tracking).

C.3 Resource Layer

A resource is a physical or virtual resource to which we wish to enable remote access via a services layer protocol. Examples of resources include compute servers (with associated queuing systems), file systems, relational databases, scientific instruments, a trouble ticket database, and wide area networks. Two other, perhaps less obvious examples of resources are a virtual organization and an advance reservation: both of which, like the other examples, have some state that XSEDE users may wish to access and manipulate.

Resources are often accessed and manipulated by local, non-standardized methods—for example, in the case of compute servers, via schedulers such as Load Sharing Facility (LSF), Torque, and Sun Grid Engine (SGE). As discussed above, the services layer both enables remote access to, and hides heterogeneities associated with, those idiosyncratic access methods. Another mechanism to mask heterogeneity is the Common User Environment (CUE).

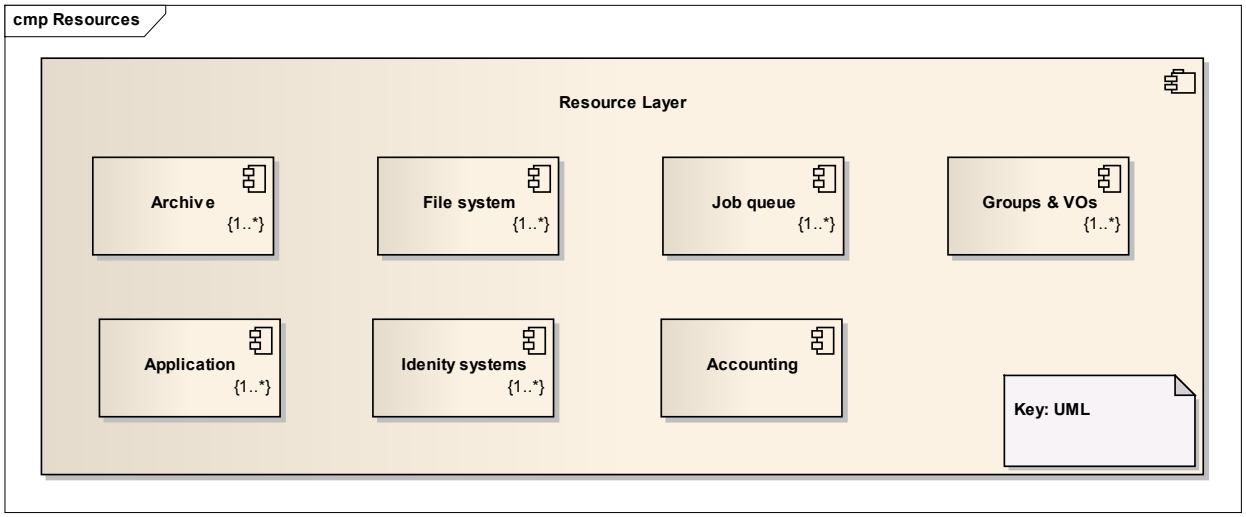

Figure 4. Examples of resources include archives, file systems, job queues, and other software and hardware systems that are virtualized by the services layer.

For example, the SGE "normal" queue on Ranger at TACC is a job queue resource. Similarly the archive (or a part of it) at the Pittsburgh Supercomputer Center is an archive resource, the Lustre file system at Indiana University is a file system resource, and the XSEDE Kerberos realm is an identity resource.

References

1. Allen, B., Bresnahan, J., Childers, L., Foster, I., Kandaswamy, G., Kettimuthu, R., Kordas, J., Link, M., Martin, S., Pickett, K. and Tuecke, S. Software as a Service for Data Scientists. *Communications of the ACM*, 55(2):81-88, 2012.
2. Bachmann, F., Foster, I., Grimshaw, A., Lifka, D., Riedel, M. and Tuecke, S. XSEDE Architecture Level 3 Decomposition, Forthcoming.
3. Demuth, B., Schuller, B., Holl, S., Daivandy, J.M. and Giesler, A., The UNICORE Rich Client: Facilitating the Automated Execution of Scientific Workflows. IEEE Sixth International Conference on eScience, 2010, 238-245.
4. Foster, I. Globus Online: Accelerating and democratizing science through cloud-based services. *IEEE Internet Computing*(May/June):70-73, 2011.
5. Goodale, T., Jha, S., Kaiser, H., Kielmann, T., Kleijer, P., Merzky, A., Shalf, J. and Smith, C. A Simple API for Grid Applications (SAGA), Version 1.1. Open Grid Forum, GFD-R-P.90, 2011.
6. Morgan, M.M. and Grimshaw, A.S., Genesis II - Standards Based Grid Computing. Seventh IEEE International Symposium on Cluster Computing and the Grid, Rio de Janeiro, Brazil, 2007, 611-618.
7. Negri, G., Shankb, J., Barberisc, D., Bosd, K., Klimentove, A. and Lamannaa, M., Distributed Computing in ATLAS. XII Advanced Computing and Analysis Techniques in Physics Research, Erice, Italy, 2008.

8. Streit, A., Bergmann, S., Breu, R., Daivandy, J., Demuth, B., Giesler, A., Hagemeier, B., Holl, S., Huber, V., Mallmann, D., Memon, A.S., Memon, M.S., Menday, R., Rambadt, M., Riedel, M., Romberg, M., Schuller, B. and Lippert, T. UNICORE 6, A European Grid Technology. Gentzsch, W., Grandinetti, L. and Joubert, G. eds. High-Speed and Large Scale Scientific Computing, IOS Press, 2009, 157-173.
9. XSEDE SYSTEM REQUIREMENTS SPECIFICATION (SRS) V3.1. XSEDE, https://www.xsede.org/c/wiki/get_page_attachment?p_l_id=57086&nodeId=51230&title=Systems+and+Software+Engineering&fileName=Systems+and+Software+Engineering%2fXSEDE+Baseline+SRS+V3.1.pdf, 2012.