GFD-E.154

Grid Storage Resource Management
https://forge.gridforum.org/projects/gsm-wg

Editors:
A. Sim
A. Shoshani
F. Donno
J. Jensen

8/18/2009

Storage Resource Manager Interface Specification V2.2 Implementations Experience Report

Status of this Document

This document provides information to the Grid community regarding the adoption of the OGF specification GFD-R-P-129 in the Storage Resource Manager Interface v2.2. It does not define any standards or technical recommendations. Distribution of this document is unlimited.

Copyright Notice

Abstract

A few groups have developed independent implementations of the Storage Resource Management (SRM) interface specification v2.2. This document describes those implementations and experiences in interoperability testing. Issues that were identified during the implementations of the specification and the production deplopyments of the implementations in various projects help develop more robust specification in the next version.

Table of Contents

1. Introduction

The Storage Resource Manager Interface Specification v2.2 (Open Grid Forum GFD-R-P-129) [http://www.ogf.org/documents/GFD.129.pdf] specifies a common control interface to storage resource management systems. Storage management is one of the most important enabling technologies for large-scale scientific investigations. Having to deal with multiple heterogeneous storage and file systems is one of the major bottlenecks in managing, replicating, and accessing files in distributed environments. Storage Resource Managers (SRMs) provide the technology needed to manage the rapidly growing distributed data volumes, as a result of faster and larger computational facilities. SRMs are Grid storage services providing not only interfaces to storage resources, but also advanced functionality such as dynamic space allocation and file management on shared storage systems [SSG03].

A Grid interface to storage is often known as a Storage Element, or SE. An SE usually provides SRM as a control interface; in addition, it may publish information in the GLUE schema, and it provides GridFTP as a common transfer protocol. Having a common schema and a common transfer protocol is an important requirement for achieving interoperability. The GLUE storage schema and GridFTP are themselves standardized in other OGF working groups. An SE providing an SRM control interface is often referred to as “an SRM”.

There are several SRM implementations for SRM servers as well as clients from independent institutions in different languages. SRM systems are deployed throughout the world, and actively used in production. This document will be of interest to anyone working with Grid storage, specifically in addressing common access to diverse storage systems with interoperating implementations.

This document describes the implementions and their interoperability tests, largely based on MSS 2007 conference paper [SHO07]. It is expected that additional documents will describe implementation aspects in more detail. Additional background information is available in GFD.129 [http://www.ogf.org/documents/GFD.129.pdf].

2. SRM Server Implementations

Over the last 8 years, there were several implementations of SRM servers. The first implementations were based on the v1.1 specifications [http://sdm.lbl.gov/srm-wg/], at several institutions in the US and Europe, including FNAL, TJNAF, LBNL, and CERN. More recently, new implementation emerged that are based on the richer v2.2 specification described in this section. We briefly describe here six such server implementations having the standard interface to a variety of storage systems. The underlying storage systems can vary from a simple disk, multiple disk pools, mass storage systems, parallel file systems, to complex multi-component multi-tiered storage systems. While the implementations use different approaches, SRM servers exhibit a uniform interface and can successfully interoperate. Short descriptions of the SRMs implementation are presented (in alphabetical order) next.

2.1. BeStMan (Berkeley Storage Manager)

BeStMan [http://sdm.lbl.gov/bestman] is a java-based SRM implementation from Lawrence Berkeley National Laboratory for disk based storage systems and mass storage systems such as HPSS [http://www.hpss-collaboration.org/hpss/index.jsp]. Its modular design allows different types of storage systems to be integrated in BeStMan while providing the same interface for the clients. Based on immediate needs, two particular storage systems are currently used. One supports multiple disks accessible from the BeStMan server, and the other is the HPSS storage system. Another storage system that was adapted with BeStMan is a legacy MSS at National Center for Atmospheric Research (NCAR) in support of the Earth System Grid (ESG) project [http://www.earthsystemgrid.org].
Figure 1 shows the design of BeStMan. The Request Queue Management accepts the incoming requests. The Local Policy Module contains the scheduling policy, garbage collection policy, etc. The Network Access Management module is responsible for accessing files using multiple transfer protocols. An in-memory database is provided for storing the activities of the server. The Request Processing module contacts the policy module to get the next request to work on. For each file request, the necessary components of the Network Access Management module and the Storage Modules (the Disk Management and the MSS Access Management modules) are invoked to process the data.

BeStMan supports space management functions and data movement functions. Users can reserve space in the preferred storage system, and move files in and out of their space. When necessary, BeStMan interacts with remote storage sites on their behalf, e.g. another gsiftp server, or another SRM. BeStMan works on top of existing disk-based Unix file system, and has been reported so far to work on file systems such as NFS, PVFS, AFS, GFS, GPFS, PNFS, HFS+, NGFS, Hadoop, XrootdFS and Lustre.

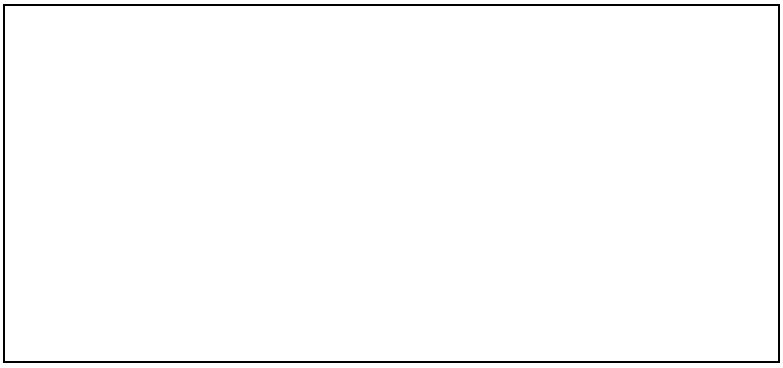

Figure 1: Overview of BeStMan architecture

2.2. CASTOR SRM

The SRM implementation for the CERN Advanced Storage system (CASTOR) [http://castor.web.cern.ch/castor/] is the result of collaboration between Rutherford Appleton Laboratory (RAL) and CERN. Like that of other implementations, the implementation faced unique challenges. These challenges were based around the fundamental design concepts under which CASTOR operates, which are different from those of other mass storage systems. CASTOR trades some flexibility for performance, and this required the SRM implementation to have some loss of flexibility, but with gains in performance.

CASTOR is designed to work with a tape back-end and is required to optimise data transfer to tape, and also to ensure that data input to front-end disk cache is as efficient as possible. It is designed to be used in cases where it is essential to accept data at the fastest possible rate and have that data securely archived. These requirements are what cause differences between the CASTOR SRM implementation and others.

The need to efficiently stream to tape and clear disk cache for new incoming data leads to two effects:

- the SURL lifetime is effectively infinite and
- the TURL, or pinning, lifetime is advisory.

In fact the latter is merely a modified garbage collection algorithm which tries to ensure those files with a low weighting are garbage collected first.

Also, space management in the CASTOR SRM is significantly different to those of other server implementations. Since the design of the MSS is to optimise moving data from disk to tape, there is no provision for allowing dynamic space allocation at a user level. The CASTOR SRM does support space reservation, but as an asynchronous process involving physical reallocation of the underlying disk servers. Other implementation designed to work with only disk based Mass Storage Systems, or a combination of disk and tape, often allow for dynamic space reservation. Another difference is that spaces are directly used to select storage with desired capabilities: for example, when a file is transferred across the WAN, it can have a space token description pointing at GridFTP servers visible to the WAN. When later the file is processed from a local cluster, it can be read with another space token description, which causes it to be copied (or staged in to) a disk pool accessible with the local protocol (RFIO).

The architecture of the CASTOR SRM, shown in Figure 2, includes two stateless processes, which interact through a RDBMS. A client-facing process (the ‘server’) directly deals with synchronous requests and stores asynchronous requests in the database for later processing. The database is therefore used to store all storage-oriented requests as well as the status of the entire system. A separate process (the ‘daemon’) faces the CASTOR backend system, and updates the status of the ongoing requests, allowing for a more fault resilient behaviour in the event the backend system shows some instability, as the clients will always be decoupled from the CASTOR backend.

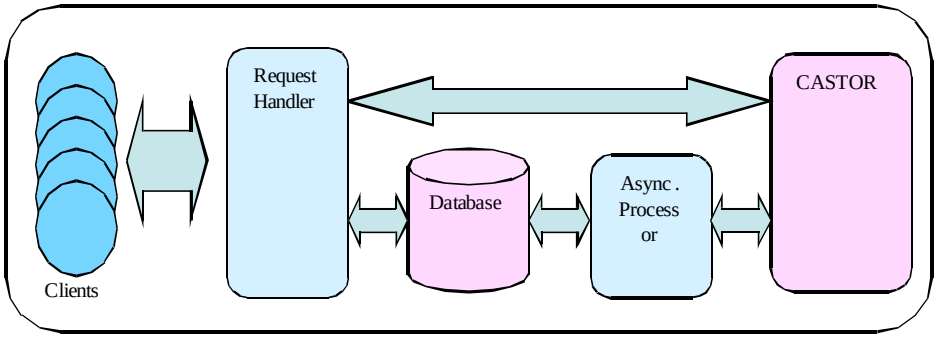

Figure 2: Overview of CASTOR SRM architecture

This architecture leverages the existing framework that has been designed and developed for the CASTOR mass storage system itself [BGL07]. The entire Entity-Relationship (E-R) schema has been designed using the UML methodology, and a customized code generation facility, maintained in the CASTOR framework, has been used to generate the C++ access layer to the database.

For data transfer and access, CASTOR supports GridFTP (via more than one implementation, some more mature than others), and the local protocol, RFIO. Newer versions of CASTOR such as v2.1.8 have improved support for xrootd.

CASTOR in production at CERN supports 21 PB on tapes and 5 PB on disks for 100M+ files.

2.3. dCache SRM

dCache [http://www.dcache.org] is a Mass Storage System developed jointly by FNAL and DESY which federates a large number of disk systems on heterogeneous server nodes to provide a storage service with a unified namespace. dCache provides multiple means of file access protocols, including FTP, Kerberos GSSFTP, GSIFTP, HTTP, and dCap and xrootd, POSIX APIs to dCache. dCache can act as a stand-alone Disk Storage System or as a front-end disk cache in a hierarchical storage system backed by a tape interface such as OSM, Enstore [http://www-ccf.fnal.gov/enstore], TSM, HPSS [http://www.hpss-collaboration.org/hpss/index.jsp], DMF or Castor

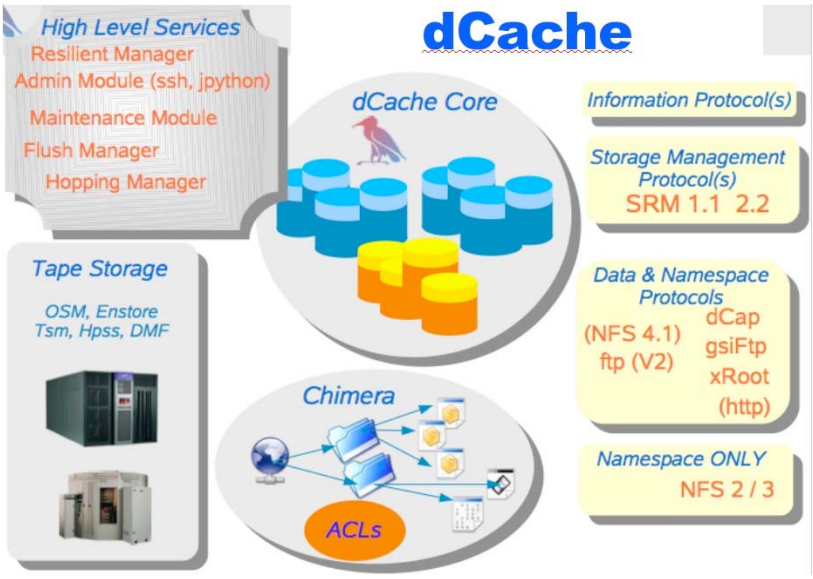

Figure 3: Overview of dCache architecture

[http://castor.web.cern.ch/castor].

dCache storage system, shown in Figure 3, has a highly scalable distributed architecture that allows easy addition of new services and data access protocols. dCache provides load balancing and replication across nodes for “hot” files, i.e. files that are accessed often. It also provides a resilient mode, which guarantees that a specific number of copies

of each file are maintained on different hardware. This mode can take advantage of otherwise unused and unreliable disk space on compute-nodes. This is a cost-effective means of storing files robustly and maintaining access to them in the face of multiple hardware failures.

The dCache Collaboration continuously improves the features and the Grid interfaces of dCache. It has delivered the gPlazma element that implements flexible Virtual-Organization (VO)-based authorization. dCache's GridFTP and GsiDCap services are implementations of the grid aware data access protocols. But the most important step to connect dCache to the Grid was the development of the SRM interface.

dCache has included an implementation of SRM Version 1.1 since 2003 and now has all protocol elements of SRM v2.2 required by the Worldwide LHC Computing Grid (WLCG) [http://lcg.web.cern.ch/LCG]. The new SRM functions include space reservation, more advanced data transfer, and new namespace and access control functions. Implementation of these features in dCache required an update of the dCache architecture and evolution of the services and core components of the dCache Storage System. Implementation of SRM space reservation led to new functionality in the Pool Manager and the development of the new Space Manager component of dCache, which is responsible for accounting, reservation and distribution of the storage space in dCache. SRM's new "Bring Online" function, which copies tape-backed files to dCache online disk, required redevelopment of the Pin Manager service, responsible for staging files from tape and keeping them on disk for the duration of the Online state. The new SRM concepts of AccessLatency and RetentionPolicy led to the definition of new dCache file attributes and new dCache code to implement these abstractions. SRM permission management functions led to the development of the Access Control List support in the new dCache namespace service, Chimera.

dCache is deployed in a large number of institutions worldwide.

2.4. Disk Pool Manager (DPM)

The Disk Pool Manager (DPM) [https://twiki.cern.ch/twiki/bin/view/LCG/DpmInformation] aims at providing a reliable and managed disk storage system for the Tier-2 sites. It is part of the Enabling Grids for E-SciencE (EGEE) project [http://www.eu-egee.org]. It currently supports only disk-based installations. The architecture is based on a database and multi-threaded daemons as shown in Figure 4.

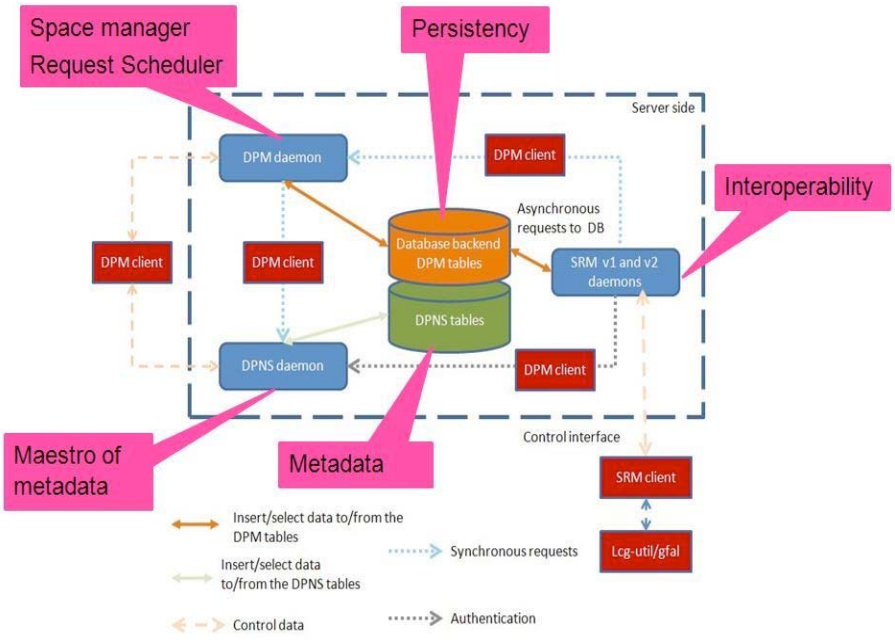

Figure 4: Overview of the DPM architecture

- The dpns daemon controls the hierarchical namespace, the file permissions and the mapping between SFN (Site File Name) and physical names; An SFN is the file path portion of an SURL.
- The dpm daemon manages the configuration of disk pools and file systems. It automatically handles the space management and the expiration time of files. It also processes the requests.
- The SRM (v1.1 and v2.2) daemons distribute the SRM requests workload (delete, put, get, etc);

- The Globus [http://www.globus.org] gsiftp daemon provides secure file transfers between the DPM disk servers and the client;
- The rfio daemon provides secure POSIX file access and manipulation.

In most cases, all the core daemons are installed on the same machine. However for large deployment, they can run on separate nodes. Although not represented in Figure 4, https and xrootd [http://xrootd.slac.stanford.edu] protocols can be used to access data.

A database backend (MySQL [http://www.mysql.com], Postgres [http://www.postgresql.org] and Oracle [http://www.oracle.com] are supported) is used as a central information repository. It contains two types of information:

- Data related to the current DPM configuration (pool and file system) and the different asynchronous requests (get and put) with their statuses. This information is accessed only by the DPM daemon. The SRM daemons only put the asynchronous requests and poll for their statuses.
- Data related to the namespace, file permissions (ACLs included) and virtual IDs, which allow a full support of the ACLs. Each user DN (Distinguished Name) or VOMS (Virtual Organization Membership Service) [https://twiki.cnaf.infn.it/cgi-bin/twiki/view/VOMS/] attribute is internally mapped to an automatically allocated virtual ID. For instance, the user Chloe Delaporte who belongs to the LHCb group could be mapped to the virtual UID 1427 and virtual GID 54. This pair is then used for a fast check of the ACLs and ownership. This part is only accessed by the DPNS daemon.

The GSI (Grid Security Infrastructure) [http://www.globus.org/security/overview.html] ensures the authentication, which is done by the first service contacted. For instance, if it is an SRM request, then the SRM daemon does the authentication. The authorization is based on VOMS.

The load balancing between the different file systems and pools is based on the round robin mechanism. Different tools have been implemented to enable users to manipulate files in a consistent way. The system is rather easy to install and to manage. Very little support is needed from the developers' team.

The DPM is currently installed at roughly 190 sites for 224 supported VOs. For a given instance, the volume of data managed ranges from a few TB up to 200 TB of data. So far no limit on the volume of data has been reported.

2.5. SRM-SRB

SRM-SRB [http://lists.grid.sinica.edu.tw/apwiki/SRM-SRB] provides the SRM interface for Storage Resource Broker (SRB) [http://www.sdsc.edu/srb/index.php] so that SRB could interoperate with other Grid Middleware and support the SRM alternative for SRB. The SRM implementation was developed at Academia Sinica, and has 3 main components as shown in Figure 5. Web service component receives the request from users and transfers the SURL into path used in the implementation. Core component gets the path from web service and communicates with the file catalog to retrieve some information like host information and file information. Core can sometimes use data server management to do operations on SRB server (or cache server). Data server management component does operations on SRB server such as getting disk usage, permission checking, etc.

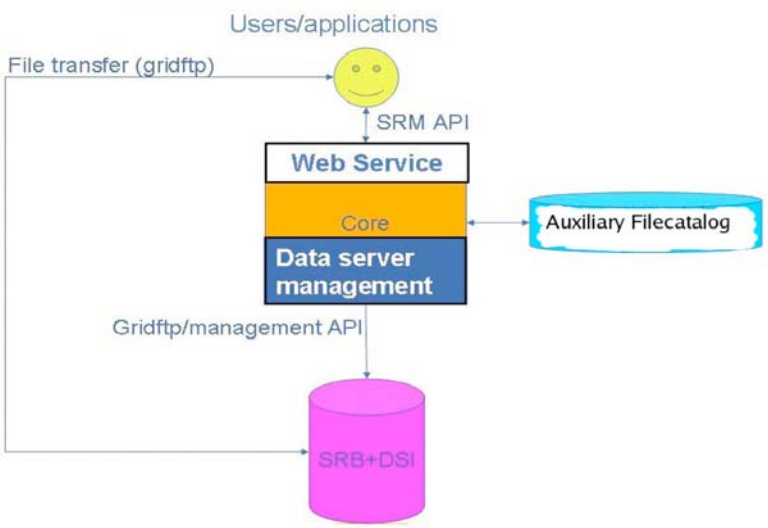

Figure 5: Overview of the SRM-SRB architecture

2.6. StoRM (Storage Resource Manager)

StoRM [http://storm.forge.cnaf.infn.it] (acronym for Storage Resource Manager) is an SRM service designed to manage file access and space allocation on high performing parallel and cluster file systems as well as on standard POSIX file systems. It provides the advanced SRM management functionalities defined by the SRM interface specification version 2.2. The StoRM project [CCD06] is the result of the collaboration between INFN – the Italian National Institute for Nuclear Physics - and the Abdus Salam ICTP for the EGRID Project for Economics and Finance research.

StoRM is designed to respond to a set of requests coming from various Grid applications allowing for standard POSIX access to files in local environment, and leveraging on the capabilities provided by modern parallel and cluster file systems such as the General Parallel File System (GPFS) [http://www-03.ibm.com/systems/clusters/software/gpfs/index.html] from IBM. The StoRM service supports guaranteed space reservation and direct access (by native POSIX I/O calls) to the storage resource, as well as supporting other standard Grid file access libraries like RFIO (Remote File I/O) and GFAL [http://www.gridpp.ac.uk/wiki/Grid_File_Access_Library].

More generally, StoRM is able to work on top of any standard POSIX file system providing ACL (Access Control List) support, like XFS and ext3. Indeed, StoRM uses the ACLs provided by the underlying file system to implement the security model, allowing both Grid and local access. StoRM supports VOMS [https://twiki.cnaf.infn.it/cgi-bin/twiki/view/VOMS/] certificates and has a flexible authorization framework based on the interaction with one or more external authorization services to verify if the user can perform the specified

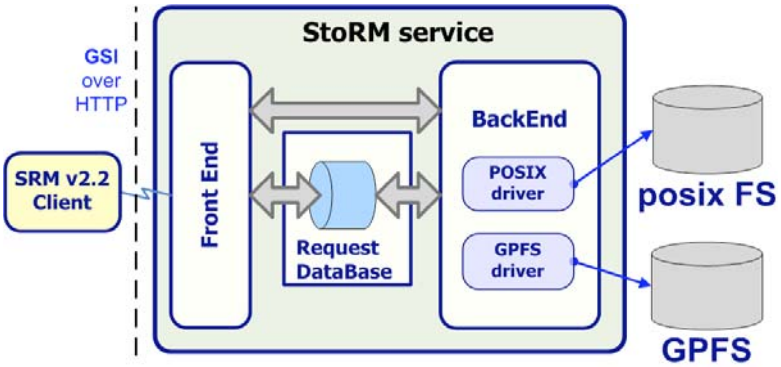

Figure 6: Overview of StoRM Architecture

operation on the requested resources.

Figure 6 shows the multilayer architecture of StoRM. The are two main components: the frontend that exposes the SRM web service interface and manages user authentication, and the backend that executes all SRM functions, manages file and space metadata, enforces authorization permissions on files, and interacts with file transfer services. StoRM can work with several underlying file systems through a plug-in mechanism that decouples the core logic from the specific file system functionalities. The specific file system driver is loaded at run time.

To satisfy the availability and scalability requirements coming from different Grid applications scenarios, one or more instances of StoRM components can be deployed on different machines using a centralized database service. Moreover, the namespace mechanism adopted by StoRM makes it unnecessary to store the physical location of every file managed in a database. The namespace is defined in an XML document that describes the different storage components managed by the service, the storage areas defined by the site administrator and the matching rules used at runtime to map the logical to physical paths. The physical location of a file can be derived from the requested SURL, the user credentials and the configuration information described in the XML document.

3. SRM Client Implementations

In this section we describe briefly some SRM clients that adhere to the SRM v2.2. While these clients use different approaches of handling user options, the SRM Clients are compatible to different SRM server implementations and successfully interoperate. In addition, two testing programs have been developed independently and are running daily to check the interoperability of these systems. Short descriptions of the SRM client implementations are presented (in alphabetical order) next.

3.1. FNAL SRM Clients

Fermi National Accelerator Laboratory (FNAL) developed a set of SRM clients [https://srm.fnal.gov/twiki/bin/view/SrmProject/SrmcpClient] in Java for selective specification interfaces. These SRM client command-lines cover most of common SRM client use cases, and include srmcp, srmmkdir, srmrmdir, srmls, srmmv, srmrm, srmping, srm-reserve-space and srm-release-space. They are compatible to all SRM servers that are available currently, and deployed worldwide.

3.2. FTS (File Transfer Service)

gLite [http://glite.web.cern.ch/glite/] File Transfer Service (FTS) [https://twiki.cern.ch/twiki/bin/view/EGEE/FTS] is a reliable data movement service that performs bulk file transfers between multiple sites of any SRM compliant storage elements. It's a multi-VO service and used to balance usage of site resources and to prevent network and storage overload.

FTS interacts with SRM in two ways, UrlCopy transfer and SrmCopy transfer. UrlCopy calls srmPrepareToGet function at the data source and srmPrepareToPut function at the data destination, and makes the 3rd party GridFTP file transfers. SrmCopy transfer uses srmCopy function. FTS is compatible to all SRM servers that are available currently, and deployed in production.

3.3. LBNL SRM Client Tools

LBNL developed SRM client tools [http://sdm.lbl.gov/wiki/Software/SRMClients/] in Java, implementing full interfaces of SRM v2.2 specification as generic SRM v2.2 clients. It consists of 34 command line clients for different functional interfaces. They are compatible to all current SRM v2.2 servers such as BeStMan, CASTOR, dCache, DPM, SRM-SRB and StoRM, and deployed worldwide. They are continuously being tested for compatibility and interoperability and optimized for performance.

3.4. Grid File Access Library and LCG Utils

To simplify user interaction with data management infrastructure, Grid File Access Library (GFAL) [http://www.gridpp.ac.uk/wiki/Grid_File_Access_Library] and LCG-Utils [http://www.gridpp.ac.uk/wiki/LCG_Utils] are developed at CERN. GFAL provides C and Python APIs for SRM functions and POSIX-like functions for creating, reading, writing, and deleting files on the Grid. LCG-Utils are high-level tools that are composed of command-lines, C and Python APIs. It hides the complexities of catalogue and

SEs interaction, and minimizes the risk of grid files corruption. It covers most of common SRM client use cases, by implementing selective specification interfaces such as srmPrepareToGet, srmPrepareToPut, srmLs, srmMkdir, srmRmdir, srmRm, srmMv, and so on, with such client tools as lcg-cp, lcg-cr, lcg-del, lcg-rep, lcg-gt and lcg-sd. LCG-Utils and GFAL are compatible to all SRM servers that are available currently, and deployed worldwide in production.

3.5. SRM Java Client Library

LBNL developed SRM Client Java API [http://sdm.lbl.gov/wiki/Software/SRMClientJavaAPI] for selective

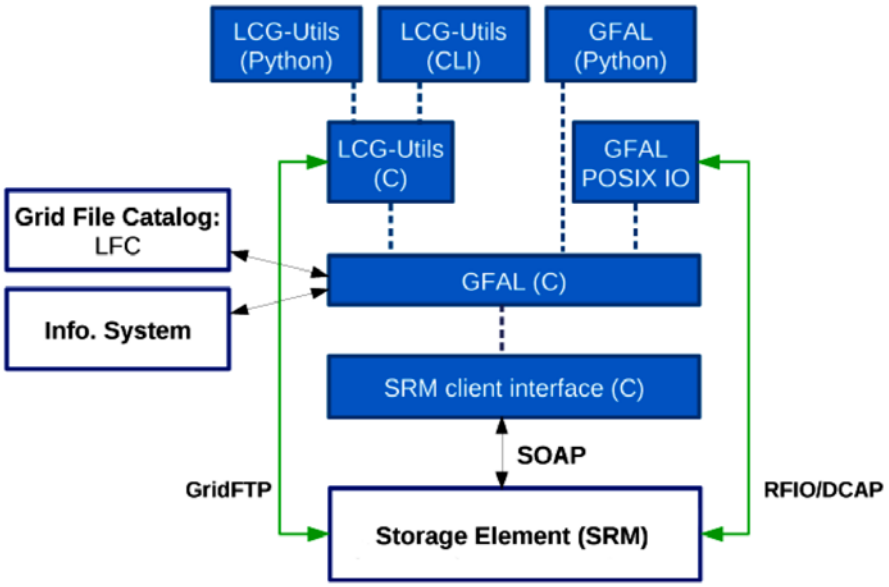

Figure 7: Overview of the GFAL and LCG-Util

specification interfaces, enabling users to take benefits from the programming their own codes. It covers most of common SRM client use cases. It is being used in Earth System Grid community and used by individuals worldwide.

3.6. S2

S2 [http://s-2.sourceforge.net] is a general-purpose test client, implementing SRM v2.2 interface specification. It is developed at CERN and RAL. S2 interprets a tree-like language that was also called S2 [http://www.livejournal.com/doc/s2/], and uses PCRE (Perl Compatible Regular Expressions) library [http://www.pcre.org]. S2 language has several attractive characteristics:

- It allows for the quick development of test programs that exercise a single test case each.
- It helps minimize human errors that are typically made in writing test cases.
- It offers an easy way to plug-in external libraries such as an SRM client implementation.
- It offers a powerful engine for parsing the output of a test, expressing the pattern to match in a compact and fully descriptive way.
- It offers a testing framework that supports the parallel execution of tests where the interactions among concurrent method invocations can be tested easily.
- It offers a "self-describing" logging facility that makes it possible to automatically publish the results of a test.

S2 tests have 5 test categories, availability tests, basic tests, cross-copy tests, use-case tests ad stress tests. It is being used in WLCG community for testing registered storage sites. These tests run daily and results are reported on the web [https://twiki.cern.ch/twiki/bin/view/SRMDev/WebHome], as shown in section 4.

3.7. SRM-Tester

LBNL developed SRM-Tester [http://sdm.lbl.gov/wiki/Software/SRMTester/] in Java, implementing full interfaces of SRM v2.2 specification. SRM-Tester has three test modes, one for manual testing for each interface, another for automated testing for all interfaces and the other for stress testing. They are compatible to all current SRM v2.2 servers such as BeStMan, CASTOR, dCache, DPM, SRM-SRB and StoRM. It is being used in Open Science Grid (OSG) [http://www.opensciencegrid.org] community for testing registered storage sites. These tests run daily and results are displayed on the web [http://datagrid.lbl.gov], as shown in section 4.

3.8. Other SRM clients

There are a few other SRM client implementations, such as BeStMan GUI SRM diagnostic tool, DPM clients, and StoRM clients. They have not yet been tested fully for interoperation among the collaboration and users. Although these SRM client implementations are available, their usage is limited so far.

4. SRM Compatibility and Interoperability Test

An important aspect in the definition of the SRM v2.2 protocol is the verification against existing implementations. The verification process has helped understanding if foreseen transactions and requirements make sense in the real world, and identifying possible ambiguities. It uncovered problematic behaviors and functional interferences in an early stage of specification development to allow for the protocol specification to be adjusted to better match existing practices. The verification process showed if the protocol adapted naturally and efficiently to existing storage solutions. In fact, it is crucial that a protocol is flexible and does not constrain the basic functionality available in existing services. As an example we can mention the time at which a SURL starts its existence in the namespace of an SRM. Server Implementations like dCache mark a file as existent in the namespace as soon as a client starts a transfer for the creation of the file. This is to avoid the need for cleanup of the name space when the client never gets to write the file. Other server implementations, instead, prefer to reserve the name space entry as soon as possible, to present a consistent view to all concurrent clients, or to simplify the interfacing with the MSS backend.

The verification process also has helped proposing and refining a conceptual model behind the protocol, with an explicit, clear and concise definition of its underlying structural and behavioral concepts. This model has made it easier to define the service semantics, helped implementation developers, and provided for a more rigorous validation of implementations. The model is a synthetic description of a user's view of the service, with the basic entities such as space and file, their relationships, and the changes they may go through. The model is described in some details in [5].

The analysis of the complexity of the SRM interface through its formal model shows that a high number of tests need to be executed in order to fully check the compliance of the server implementations to the specifications. Therefore, an appropriate testing strategy has to be adopted in order to reduce the number of tests to be performed to a manageable level, while at the same time covering those aspects that are deemed to matter in practice.

Testing activities aim at finding differences between the actual and the intended behavior of a system. In particular, [MSB04] gives the following definition: "Testing is the process of executing a program with the intent of finding errors." A test set is defined to be exhaustive if and only if it fully describes the expected semantics of the specifications, including valid and invalid behaviors.

In order to verify the compliance to a specification of available implementations of SRM v2.2, 5 categories of tests have been designed. Furthermore, many hypotheses have been made in order to make the model simpler and to reduce the total number of tests, while keeping the test sets valid and unbiased. The 5 families of tests are the following:

- Availability: the srmPing function and a full put cycle for a file are exercised (srmPrepareToPut, srmStatusOfPutRequest, file transfer, srmPutDone). This family is used to verify availability and very basic functionality of an SRM endpoint.
- Basic: the equivalence partitioning and boundary condition analysis is applied to verify that an implementation satisfies the specification when it has a single SRM call active at any given time. Basic tests, in turn, can be split into categories that depend only on each other and on earlier groups:
  - basic information (srmPing, srmLs, srmGetTransferProtocols),
  - permissions,

- directory functions,
- advanced permissions and directory functions,
- recursive ls,
- the put operations cycle (srmPrepareToPut, etc.),
- file management (including srmPrepareToGet),
- srmCopy,
- request suspension,
- basic space functions,
- advanced space functions.

- Use cases: cause-effect graphing, exceptions, functional interference, and use cases extracted from the middleware and user applications are exercised.
- Interoperability: remote operations (servers acting as clients for some basic SRM functions) and cross copy operations among several implementations are executed.
- Stress: the error guessing technique and typical stress situations are applied to verify resilience to load.

The S2 families of tests run automatically 5 times a day. The results of the tests are published on a web page [https://twiki.cern.ch/twiki/bin/view/SRMDev/WebHome]. In particular, the data of the last run together with the history of the results and their details are stored and made available to the developers through the web. Plots can be produced every month on the entire period of testing to track the improvements and detect possible problems.

The testbed that we set up includes five different implementations: CASTOR, dCache, DPM, BeStMan, and StoRM. It currently has 23 available endpoints located in Europe and the US. In particular, 5 endpoints are where the main development happens. These endpoints have been tested for over a year and a half. The other endpoints have been added recently. They are used to verify that the implementation can accommodate different specific needs at different sites and help smooth the installation and configuration process.

The test suite is built as a perl wrapper gluing all of the 36 individual test modules, corresponding almost one to one to the 38 SRM v2.2 interfaces. Each test module is a small C application, and is built on top of gSOAP 2.6. It was written mainly to allow DPM srmv2.2 implementation, but has also been used to crosscheck some features of BeStMan and dCache SRM v2.2 front-ends. It is most of the time used as a regression test to ease the development lifecycle, and new use cases and specific tests are added as soon as new features become available on the DPM SRM v2.2 server. It now includes about 400 different steps, and runs in about 500 sec. Transfers are achieved through Secure RFIO or GridFTP when testing a DPM server, but are switched back to GridFTP only when testing some other server. In Figure 8, the summary availability test [http://lxdev25.cern.ch/s2test/avail/s2_logs/] is shown. Figures 9 shows the summary of basic [http://lxdev25.cern.ch/s2test/basic/s2_logs/] and use case [http://lxdev25.cern.ch/s2test/usecase/s2_logs/] tests. Figure 10 shows the summary of cross-copy test results [http://lxdev25.cern.ch/s2test/cross/s2_logs/]. While for the basic and use case families of tests the errors have improved greatly in a relatively short time, we still have to do some work in terms of interoperability and cross copy operations. Stress testing [https://twiki.cern.ch/twiki/pub/SRMDev/WebHome/StressTest_Description.txt] has just started and some of the available endpoints are being equipped with more resources for that. The instabilities shown in the results usually are caused by service upgrades (to deploy fixes in the code) or circumstances where the server is too busy serving other requests (when the endpoint is a production system not dedicated to tests). Also,

Summary of S2 SRM v2.2 availability test - Tuesday 13 January 2009 06:37pm CET

CERN ALICE	ASGC C2	CERN ATLAS	CERN C2	CERN CMS	CNAF C2	CERN DTEAM	CERN LHCb	CERN PPS	BNL PROD dCache	DESY dCache	FZK PROD dCache	IN2P3 PROD dCache	NDGF PROD dCache	PIC PROD dCache	SARA PROD dCache	FNAL dCache	TRIUMF PROD dCache	CERN DPM	LAL DPM	NIKHEF DPM	LBNL BeStMan	CNAF StoRM2
UP	DOWN	UP	DOWN	UP	UP	UP	UP	UP	UP	DOWN	UP	UP	UP	UP	UP	UP	UP	UP	UP	UP	UP	UP

Figure 8: Sample screen of S2 availability test results

underpowered hardware can limit the transaction rates.

Figure 9: Sample screen of S2 basic and use-case test results

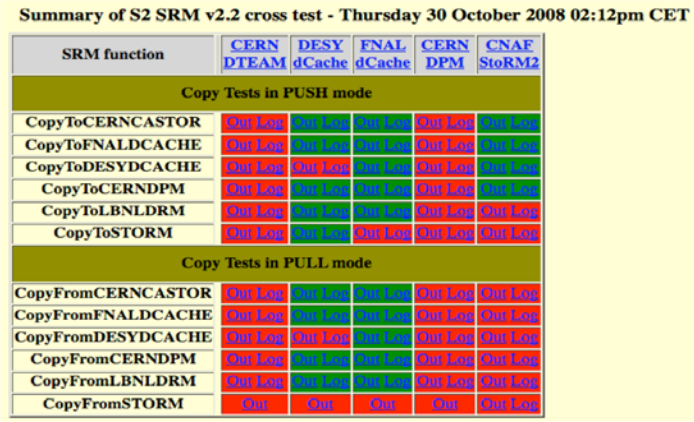

Figure 10: Sample screen of S2 cross-copy test results

Another SRM test program [http://sdm.lbl.gov/wiki/Software/SRMTester/] was developed at LBNL, and is being run several times daily, and the results published [http://datagrid.lbl.gov]. The testbed includes five different server implementations: BeStMan, CASTOR, dCache, DPM, and StoRM. It currently has 24 available endpoints located in the US and Europe. In particular, 5 endpoints are the main developmental endpoints. These endpoints have been tested for over a year and a half. The other endpoints have been added recently.

In Figure 11, the summary of the test results [http://datagrid.lbl.gov/v22/] on testing sites is shown on the left figure, and each functional testing result on different dates is shown on the right figure. Figures 12 shows the functional testing results on the developmental endpoints [http://datagrid.lbl.gov/v22/index-dev.html].

S2 and SRM-Tester compliment each other in that S2 uses C/C++ clients while SRM-Tester uses java clients.

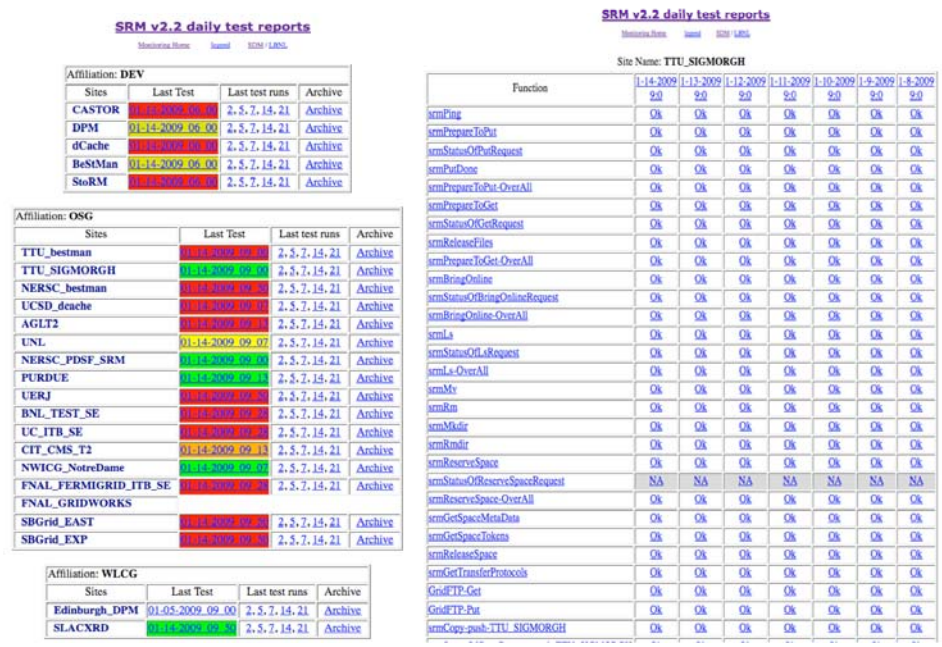

Figure 11: Sample screen of SRM-Tester test results

4.1. I

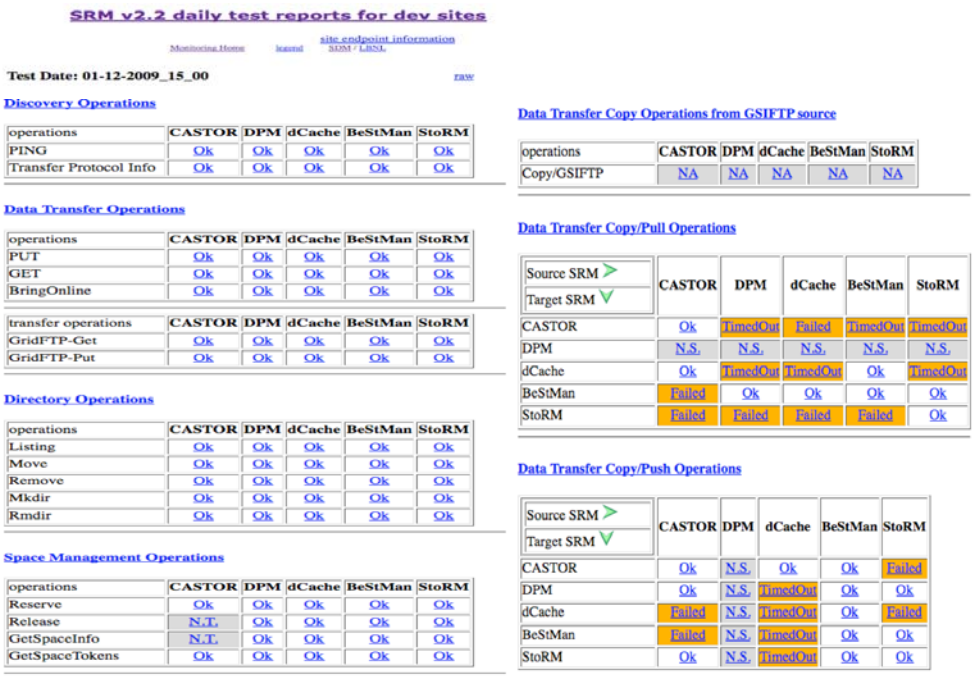

Figure 12: Sample screen of SRM-Tester test results

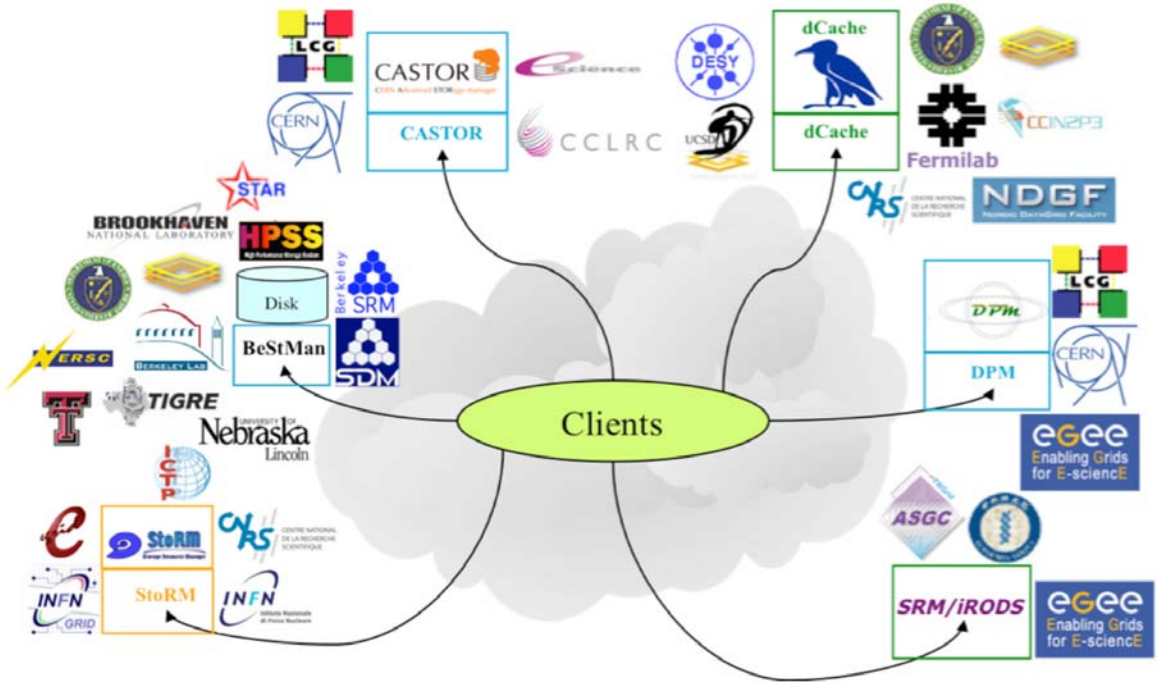

Figure 13: SC2008 Demo – Interoperability of 6 SRM server implementations at 12 participating sites

In the Super Computing 2007 and 2008 conferences, SRM implementations successfully demonstrated its compatibility and interoperability [https://sdm.lbl.gov/bestman/docs/sc08/SC08-SRM.html]. In collaboration with OGF GIN-WG, WLCG, EGEE, OSG and ESG, we have had multiple sites participating with all currently implemented SRMs. Figure 13 shows the diagram of the demonstration.

4.2. Implementation-dependent differences

Despite the uniform interfaces, implementation characteristics due to the underlying storage system occasionally cause confusion to SRM clients and their usage as users sometimes have to take into account at runtime which storage system at which site they are communicating with. They arise when the underlying storage system has either different aspects of storage managements that do not map directly to the abstract SRM concepts and interfaces, or different operational environements that cannot be mapped or exposed to the interfaces. One example is when srmCopy is supported only in PUSH mode due to the design of the underlying storage system. Clients should know this information for successful requests before they make an srmCopy request. Recommended way for unsupported behavior is to handle SRM_NOT_SUPPORTED message as the return status from the server. This is in practice used in most of SRM implementations.

5. SRM Deployments

Through LCG [http://lcg.web.cern.ch/LCG/] and EGEE [http://www.eu-egee.org/] projects, more than 250 SRM deployments are in production in Europe, Asia, Canada, South America, Australia and Africa, managing more than 10PB (as of 11 Nov. 2008). There are roughly 194 DPM SRMs at 185 sites, 57 dCache SRMs at 45 sites, 6 CASTOR SRMs at 6 sites, 22 StoRM SRMs at 21 sites, and 1 SRM-SRB at 1 site, which is a relatively new implementation.
In the US, about 70 SRM deployments of dCache SRM and BeStMan SRM are estimated in production through OSG [http://www.opensciencegrid.org], ESG [http://www.earthsystemgrid.org] and other projects (as of 30 Apr. 2009).

6. SRM v2.2 specification issues and future directions

During the extensive work on various SRM implementations and deployments, we have noticed a number of issues. They have been collected and discussed [https://twiki.cern.ch/twiki/bin/view/SRMDev/WebHome] in the group.

Issues were divided into those that could have been fixed and resolved for minor editorial changes and other behavioral clarifications, and those feature requests that require new functionalities. Issues that require new functionalities have been postponed for the next version of the specification. Some of the discussed issues are listed in Appendix A.
Next version of SRM interface specification has been in discussion. The next version would accommodate issues that have not been resolved so far and suggestions and features that would promise better storage interface on the grid.
Other informational documents on topics such as the programmable details of an implementation and SRM testing details on compatibility and interoperation are in progress as separate documents.

7. Conclusion

We have described the international collaboration that led to the Storage Resource Manager (SRM) protocol standard, multiple implementations based on this standard, and the validation processes for checking adherence to the SRM protocol. The key reasons for the success of adaptation of the SRM standard are: (a) an open protocol, unencumbered by patents or licensing, (b) an open collaboration where any institution willing to contribute can join, and (c) a well establish validation process. We have described five interoperating implementations, many of which are open source. We have described how the SRM provides a standard interface to diverse storage systems to the Grid, including file systems that support a single disk or disk pools, distributed file systems, as well as hierarchical mass storage systems.

The SRM protocol supports advanced capabilities, such as dynamic space reservation, that enables advanced Grid clients to make use of these capabilities. However, since storage systems are diverse, implementation of such advanced capabilities is optional. On the Grid, SRM is the basis for the Storage Element (SE) specification in the widely-used GLUE information schema, which allows clients to discover services supporting the desired capabilities.

In this document, we also described our collaboration in developing and using test tools that have been crucial to the validation and the interoperability of the implementations. A large range of tests was developed and used over multiple implementations, from testing the correctness of individual functions in the API to testing complex use cases and control flow. The testing tools helped in not only in discovering interoperability before the users do, thus leading to improved experience of the SRM services in the users' view, but also allowed advanced optional features to be tested incrementally as they become supported by each implementation.

8. Security Considerations

Security Issues are outside the scope of this document. For security considerations of the SRM specifrication, please refer to the GRD-R-P.129 document.

9. Contributors

9.1. Editors information

Alex Sim
Lawrence Berkeley National Laboratory
1 Cyclotron Road, MS 50B-3238
Berkeley, CA 94720, USA
Email: ASim@lbl.gov

Arie Shoshani
Lawrence Berkeley National Laboratory
1 Cyclotron Road, MS 50B-3238
Berkeley, CA 94720, USA
Email: Shoshani@lbl.gov

Jens Jensen
Rutherford Appleton Laboratory
Harwell Science and Innovation Campus

Oxon, OX11 0QX, UK
Email: jens.jensen@stfc.ac.uk

Flavia Donno
European Organization for Nuclear Research (CERN)
Geneva, Switzerland
Email: Flavia.Donno@cern.ch

9.2. Contributors

Authors of the SRM implementations are people who have been involved in SRM over the years. The list includes, by institution:

Lana Abadie, Paolo Badino, Jean-Philippe Baud, Flavia Donno, Akos Frohner, Birger Koblitz, Sophie Lemaitre, Giuseppe Lo Presti, Maarten Litmaath, Rémi Mollon, David Smith, Paolo Tedesco
European Organization for Nuclear Research (CERN) Switzerland

Patrick Fuhrmann, Tigran Mkrtchan
Deutsches Elektronen-Synchrotron (DESY) Germany

Matt Crawford, Dmitry Litvinsev, Alexander Moibenko, Gene Oleynik, Timur Perelmutov, Don Petravick
Fermi National Accelerator Laboratory (FNAL) USA

Ezio Corso, Massimo Sponza
International Centre for Theoretical Physics (ICTO) Italy

Alberto Forti, Luca Magnoni , Luca Magnoni, Riccardo Zappi
Istituto Nazionale di Fisica Nucleare (INFN) Italy

Gilbert Grosdidier
LAL / IN2P3 / CNRS, Faculté des Sciences, Orsay Cedex, France

Junmin Gu, Vijaya Natarajan, Arie Shoshani, Alex Sim
Lawrence Berkeley National Laboratory (LBNL) USA

Shaun De Witt, Jens Jensen
Rutherford Appleton Laboratory (RAL) UK

9.3. Acknowledgement

This document preparation has been partially supported by the Office of Energy Research, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. This work is also co-funded by the European Commission through the EGEE project under contract number INFSO-RI-031688.

10. Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to

identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

11. Disclaimer

This document and the information contained herein is provided on an "As Is" basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

12. Full Copyright Notice

13. References

[BGL07] O. Bärring, R. Garcia Rioja, G. Lo Presti, S. Ponce, G. Taurelli, D. Waldron, *CASTOR2: design and development of a scalable architecture for a hierarchical storage system at CERN*, CHEP, 2007
[CCD06] Corso, E. and Cozzini, S. and Donno, F. and Ghiselli, A. and Magnoni,, L. and Mazzucato, M. and Murri, R. and Ricci, P.P. and Stockinger, H. and Terpin, A. and Vagnoni, V. and Zappi, R., *StoRM, an SRM Implementation for LHC Analysis Farms Computing in High Energy Physics*, CHEP, 2006
[DD07] A. Domenici, F. Donno, *A Model for the Storage Resource Manager*, Int. Symposium on Grid Computing 2007
[MSB04] G. J. Myers, C. Sandler (Revised by), T. Badgett (Revised by), T. M. Thomas (Revised by) The ART of SOFTWARE TESTING 2 edition, December 2004
[SHO07] Arie Shoshani et al, *Storage Resource Managers: Recent International Experience on Requirements and Multiple Co-Operating Implementations*, 24th IEEE Conference on Mass Storage Systems and Technologies (MSST 2007), September 2007, San Diego, California, USA. IEEE Computer Society 2007
[SSG03] A. Shoshani, A. Sim, and . Gu, *Storage Resource Managers: Essential Components for the Grid, in Grid Resource Management: State of the Art and Future Trends*, Edited by Jarek Nabrzyski, Jennifer M. Schopf, Jan weglarz, Kluwer Academic Publishers, 2003

14. Appendix A

Issues have been collected and discussed [https://twiki.cern.ch/twiki/bin/view/SRMDev/WebHome] in the group. Issues were divided into those that could have been fixed and resolved for minor editorial changes and other behavioral clarifications in the v2.2 specification, and those feature requests that require new functionalities. They have been postponed for the next version of the specification.

Here's a partial list of the discussed issues.

14.1 Fixed Issues

These issues have been discussed and fixed in the current specification.

- srmLs can return SRM_REQUEST_QUEUED at file level.
- Asynchronous calls can return REQUEST_INPROGRESS; srmLs at the request level.
- On page 54, point h) is empty: It is removed.
- srmStatusOfPutRequest should return SRM_ABORTED after a successful srmAbortRequest has been executed. That is the same case for srmStatusOfGetRequest.
- Some status methods for asynchronous SRM functions should return REQUEST_INPROGRESS:
  - srmStatusOfLsRequest (both at request and file level)
  - srmReserveSpace (at request level)
- There is reference to "volatile, durable, permanent" space, even though those attributes are never defined. See, for instance, srmReleaseSpace.
  - 2.3.2.b in the spec is changed to OUTPUT or CUSTODIAL retention quality space, instead of durable or permanent space.
- The specification does not state if token descriptions are case sensitive or not. Different implementations behave differently.
  - The spec is changed for v2.2 to state that token descriptions are case sensitive and immutable.
- On page 52 point h) and on page 58 point i) of the specification, the method srmGetRequestID is mentioned instead of srmGetRequestTokens.
- In srmUpdateSpace, the output parameter lifetimeGranted is the lifetime left relative to the request calling time.
- The space token is mandatory in srmExtendFileLifeTimeInSpace. Paragraph 2.9.2a is fixed.
- On page 16 of the specification, section 1.28, second bullet stated that the data structure TGetFileRequest contains the TAccessPattern. This is not the case. The statement should be replaced with something that states that TTransferParameters can be specified during an srmPrepareToGet, srmPrepareToPut, srmBringOnline reqest. What specified in such data structure might collide with the characteristics of the space specified, and in this case TTransferParameters must be ignored.
- In srmReserveSpace the desiredLifeTime is an unsigned long, while in all other SRM methods it is an int.
  - It should be int, and fixed.
- In srmAbortRequest, note a. states that "Expired files are released.". This is not an effect of the srmAbortRequest. This sentence must be removed.

14.2 Deferred Issues

These issues have been discussed, and the discussion and/or the resolution have been deferred to the next version of the specification due to the new functional requirements or production deployment schedules.

- SRM_DONE is not used and should be removed from the specification.
- No methods allow getting the associated description (if it exists), given a space or request token. This requires a change in the web services interface.
- No methods allow for the retrieval of all space token descriptions for a given VO or FQAN. This requires a change in the web services interface.
- No methods allow for the listing of the content of a space, associated with a space token.
- After a put operation, in the space there is a copy of the file even if the handle (TURL) is expired or gone. What is its lifetime? This has an implication on srmReleaseSpace of that space.
  - A copy of the file stays in space and its lifetime equals the SURL lifetime.
- The function of the paramenter overwriteOption in an srmPrepareToPut should be clarified. It is valid at a request level, but it does not have any permission meaning. It is the equivalent of the -f (force) option in the UNIX commands cp.

14.3 Clarified Issues

These issues have been discussed and clarified in the current v2.2 specification.

- srmExtendFileLifeTime has an ambiguity and allows for the following 2 possibilities in case the newTimeExtended exceeds the remaining lifetime of the space:

    - Return PARTIAL_SUCCESS at the request level and SRM_FAILURE at the file level, and the TSURLLifetimeReturnStatus returns the remaining lifetime.
    - SUCCESS at the request and file level and TSURLLifetimeReturnStatus contains the remaining lifetime. Case b is the correct one, and should be specified in the specification.
- Specify what values lifetimeLeft and lifetimeAssigned are assumed in TMetaDataPathDetail and in TMetaDataSpace, after the execution of a Ls. Are 0 and -1 allowed values? What is their meaning? Same for remainingPinLifetime and remainingFileLifetime in TPutRequestFileStatus, TCopyRequestFileStatus, TSURLLifetimeReturnStatus and everytime the Lifetime appears as an output paramenter; srmReserveSpace, srmUpdateSpace, srmStatusOfUpdateSpaceRequest, srmStatusOfReserveSpaceRequest, srmPrepareToGet, srmStatusOfPrepareToGetRequest
- Default value for the parameter overwriteOption in an srmPrepareToPut request is up to the implementation.
- It should be clarified the behavior after an srmAbortRequest is issued.
    - If all the files are rather completed or aborted, this request should return SUCCESS: The server correctly cleaned up the request, and there's nothing more that the client should do to release the resources associated with the request. Eventually, the client has to call srmRm to delete the completed files (SURLs).
- It should be explicitly stated that in srmExtendFileLifeTime a request that specifies both file and pin lifetime is invalid.
- The behavior of srmAbortRequest on a copy request should be clarified. In copy cases, either source or target has to be local to the SRM server. There can be one source SURL and multiple target SURLs in a request, regardless of PULL or PUSH mode. The question is how clients abort the request; by the file, by the source or the target? If clients abort by the source SURLs, all file transfer of the same source SURL will be aborted. If clients abort by the target SURL, only the particular target file operation will be aborted, and others from the same source will not be aborted. Therefore either matching case will be honored.
- If a particular combination of Retention Policy and Access Latency is not supported in srmReserveSpace the SRM server must return SRM_NOT_SUPPORTED
- A general behaviour of all srm interfaces is that if a particular SRM server does not support a particular optional parameter, SRM_NOT_SUPPORTED must be returned.
- srmAbortFiles returns SRM_ABORTED at file level in some implementation while only SRM_SUCCESS, SRM_INVALID_PATH and SRM_FAILURE are possible.
    - At file level the interface should return SRM_SUCCESS and not SRM_ABORTED.
- srmReleaseFiles is used to reset the TURL lifetime to 0. Therefore it can be performed on files resulting from srmPrepareToGet, srmBringOnline, srmPrepareToPut and srmPutDone. srmReleaseFiles is not allowed after an srmCopy since there is not directly exposed TURLS associated with the file. Does this mean that files are released after an srmPutDone in a srmCopy operation in order to avoid that for the implementations honoring pins the file stays around in some space with some lifetime?
    - srmReleaseFiles is only valid after srmPrepareToGet or srmBringOnline operation. To release a TURL after srmPrepareToPut, srmAbortFiles or srmAbortRequest should be used. If a client issues srmReleaseFiles after an srmPrepareToPut or srmPutDone, then the SRM server must return SRM_INVALID_REQUEST.
- The TOverwrite Mode WHEN_FILES_ARE_DIFFERENT is not supported by any implementations. What is the behavior foreseen?
    - This functionality is not implemented at the moment by any implementations. Files can be different when the declared size, checksum or any other properties for a SURL differs from the actual one.
- In srmReleaseFiles, if the request token is not provided, then authorizationID is needed. It is not clear what this field is for. It looks like both SURL and the request token can be left unspecified.
    - When request token is provided and no SURLs are provided, all files that belong to the request (that is associated with the request token) will be released. When request token is not provided and SURLs are provided, those SURLs that belong to the client ID will be released. But in this case, some verification is needed that those SURLs belong to the client ID, and that can be done with the authorizationID. There cannot be the case that none of SURLs and request token is provided, and at least one of requestToken and SURLs must be provided. AuthorizationID is

additional information that can verify the client, and it is useful specially when everyone is mapped into one login in the grid-mapfile.

- srmExtendFileLifeTime behaviour:
  - The interface allows changing only one type of lifetime at a time (either SURL lifetime or pin lifetime), depending on the presence or absence of the request token.
  - When SURL lifetime is extended with newFileLifetime, the request token must not be specified.
  - When the request token is specified, pin lifetime is always extended.

  When lifetime input parameters are not specified, the SRM server applies the default values. (-1 and 0 lifetime have the same interpretation as before; indefinite and default)
- srmExtendFileLifeTime[InSpace] has an ambiguity and allows for the following 2 possibilities in case the newTimeExtended exceeds the remaining lifetime of the space:
  - (a) Return PARTIAL_SUCCESS at the request level and SRM_FAILURE at the file level and the TSURLLifetimeReturnStatus returns the remaining lifetime.
  - (b) SUCCESS at the request and file level and TSURLLifetimeReturnStatus contains the remaining lifetime.
  - Case (b) from above is the correct one and should be specified in the specification.
- Streaming mode is allowed on srmPrepareToGet, srmPrepareToPut, srmCopy and srmBringOnline operations. If streaming mode is supported, in case there is not enough space, the SRM server returns SRM_REQUEST_QUEUED, and keeps trying for the duration of desiredTotalRequestTime specified by the client. In the explanation field for the request, the SRM server can make explicit the status that a retry is ongoing. If instead a SRM server does not support streaming mode, then at file level the errors SRM_NO_USER_SPACE (if the client has specified a space token on srmPrepareToPut) or SRM_NO_FREE_SPACE can be returned and the request return code can be either SRM_PARTIAL_SUCCESS (if some files were successful) or SRM_FAILURE.
- Specify estimatedWaitTime = -1 to mean unknown value.
- After an srmPrepareToPut, a client is not forced to call srmPutDone nor an srmAbortRequest in order to set to 0 the pintime of the TURL. Therefore the pintime of the TURL passed as an input argument of the interface is used to make the TURL expire, in case the client does not issue srmPutDone or srmAbortRequest.
- After an srmPrepareToPut, the SURL exists but it is marked as SRM_FILE_BUSY in an srmLs operation. When a SURL is in this status, its lifetime cannot be extended.
- In the interface srmExtendFileLifeTimeInSpace, arrayOfSURLs is optional. This means the new lifetime must be applied to all files in the space if arrayOfSURLs is NULL.
- In the interface srmExtendFileLifeTimeInSpace, the pinLifeTime field in the returned TSURLLifetimeReturnStatus will be left NULL, since the interface only applies to SURLs.
- srmExtendFileLifeTime can be used to extend the lifetime of a SURL (the result of srmCopy or srmPutDone operation). It can be used to extend the pinLifetime of valid TURLs (result of srmPrepareToPut, srmPrepareToGet). It cannot be used to extend the pinLifeTime after an srmCopy or srmPutDone operation is completed. In such case, the interface should return SRM_INVALID_REQUEST at the file level and SRM_PARTIAL_SUCCESS or SRM_FAILURE at the request level. Furthermore, lifetime for SURLs resulting from an srmPrepareToPut can be extended through this interface.
- For SURLs on which an srmPrepareToPut is being operated, an srmLs, srmBringOnline, srmPrepareToGet will return SRM_FILE_BUSY, a second srmPrepareToPut or srmCopy will return SRM_FILE_BUSY if the SURL can be overwritten, otherwise they return SRM_DUPLICATION_ERROR.
- Lifetime of SURLs only start when srmPutDone is executed.
- srmSetPermission and srmRm can be invoked even when the status of the SURL is SRM_FILE_BUSY.
- srmMv cannot be operated when the status of the SURL is SRM_FILE_BUSY. The return code must be SRM_INVALID_REQUEST.
- srmChangeSpaceForFiles cannot be operated when the status of the SURL is SRM_FILE_BUSY. In this case the return code at file level must be SRM_INVALID_REQUEST.
- If srmReleaseFiles is requested after an srmPrepareToPut or an srmPutDone request, the request fails with SRM_INVALID_REQUEST.
- If a SURL is being written (by a put or copy operation) and in the mean time an srmRm gets executed successfully, the subsequent srmPutDone will return SRM_INVALID_PATH at the file level.

- If two srmPutDone are executed for the same request for the same SURL, the second gets SRM_DUPLICATION_ERROR at the file level and SRM_SUCCESS or SRM_PARTIAL_SUCCESS or SRM_FAILURE at the request level.
- srmReleaseSpace is allowed on a space where a SURL is being created. If file is volatile and forceFileRelease has been set to true, the file is removed and SRM_INVALID_PATH returned by the srmPutDone at file level. If the file is permanent, the file is moved in the default space and the space is successfully released (if no other pinned files are there). The subsequent srmPutDone would be successful. If forceFileRelease is set to false, the srmReleaseSpace should fail and return SRM_FAILURE.