WLCG Addendum to the SRM v2.2 Memorandum of Understanding

To: CCRC'08 Storage Solutions Working Group

From: Flavia Donno

Date: 4/14/2008

Version: 0.3

Comments: A. Frohner, G. Oleynik, D. Smith, T. Perelmutov

Purpose of this document

This note summarizes the agreed client usage and server behavior for the SRM v2.2 implementations used by WLCG applications. The agreement encompasses the clients:

- FTS
- GFAL
- lcg-utils
- dCache srm clients

and storage providers:

- CASTOR
- dCache
- DPM
- StoRM

The agreement shall be focused on meeting the LHC experiments' requirements for Grid storage interfaces. The note specifies how the various existing methods and data objects, as they are specified in the agreed v2.2 specification [1], should be interpreted and/or extended in order to meet the LHC requirements. The outlined WLCG interpretation of the interfaces may suggest sometimes a more restrictive or extended use of the specification than other implementations of SRM client and servers although care is taken to preserve interoperability with the latter.

The document also specifies what is required in the medium-term (May 2008) and what is required in the long-term.

This document is open for discussions/corrections/comments/addition.

1. WLCG SRM data model interpretation agreement

In what follows, we detail the interpretation of the SRM v2.2 concepts that were found to be controversial during the experience acquired operating an SRM v2.2 based Grid infrastructure.

1.1 The SRM space

Definition: An SRM space is a logical view of an ***online*** physical space allocation that is reserved for read/write operations on files.

An SRM space is characterized by several properties:

- Retention Policy Information (Retention Policy and Access Latency)
- Owner
- Connection Type (WAN , LAN)
- Supported Operations (READ, WRITE, READ-WRITE)
- Supported File Access/Transfer Protocols
- Space Token
- Space Token Description (optional)
- Status
- Total Size
- Guaranteed Size
- Unused Size
- Assigned Lifetime
- Left Lifetime
- Client Networks

In WLCG spaces are statically reserved, although support for truly dynamic space reservation can be provided by some implementations.

When a file is removed from a space, the space occupied by the file is returned back to the original space allocation.

When a file is copied to tape and purged from disk, the space occupied by the file is returned back to the original space allocation.

Only the owner of a space or the resource administrator can release the space.

The spaces can be read/write or read-write, writing into a read/stage space might have no meaning as well as trying to read from a write only space can be impossible.

1.2 SRM files and copies

Definition: A file is a set of data with the properties defined on pag. 19 of [1], paragraph 2.25:

```
typedef   struct {string                path,   // absolute dir and file path
                  TReturnStatus         status,
                  unsigned long         size,    // 0 if directory
                  dateTime              createdAtTime,
                  dateTime              lastModificationTime,
```

```
                TFileStorageType            fileStorageType,
                TRetentionPolicyInfo        retentionPolicyInfo,
                TFileLocality               fileLocality,
                string[]                    arrayOfSpaceTokens,
                TFileType                   type,       // Directory or File
                int                         lifetimeAssigned,
                int                         lifetimeLeft,     // on the SURL
                TUserPermission             ownerPermission,
                TGroupPermission            groupPermission,
                TPermissionMode             otherPermission,
                string                      checkSumType,
                string                      checkSumValue,
                TMetaDataPathDetail[]       arrayOfSubPaths      // optional recursive
} TMetaDataPathDetail
```

A file is furthermore characterized by a Site URL and by the space the file was first created into. This space is referred to as the *file master space* that determines the characteristics of the file.

A file can have several copies in several spaces.

Definition: A ***copy*** of a file is a physical instance of the file in a given space. It is characterized ~~by the RequestID of the srm request that has generated the copy and~~ by the properties defined on pag. 19 of [1]. (The first copy of a file is referred to as the *file primary copy*.)

A copy of a file might have an associated transport URL (TURL). Such TURL has an associated pin-lifetime. Multiple TURLs may refer to the same copy of the same file or to different physical copies of the same file. A TURL is a user –guaranteed handle to the file. It is valid till its lifetime or associated pin is not expired. If one of the TURLs associated to a copy of a file is released, the other TURLs whose lifetime have not yet expired will continue to stay valid. A TURL is generated by the system after an srmPrepareToGet or srmPrepareToPut request and might be associated to the request that has generated it.

~~1.3 SRM handles~~

~~*Definition*: A SRM handle is a user-guaranteed transport-specific URL (TURL) with associated pin-lifetime and an online copy (existing or to be created) of a file in a space. A handle is always associated to an SRM request. This expresses the characteristics of the space, a user-created one or the default, where the copy of the file associated to the corresponding TURLs lays. A file handle is characterized by the following properties:~~

- ~~A string defining the file handle or TURL~~
- ~~A handle lifetime~~
- ~~The request ID of the SRM request generating the TURL. Possible requests are srmPrepareToPut or srmPrepareToGet.~~

2. WLCG requirements

In this section we describe the experiment requirements that took to the definition of this Addendum to the SRM v2.2 WLCG Mou. The requirements are listed in order of the priority given by the LHC experiments, from the most important and more urgent to implement to the less urgent.

A. ***Protecting spaces from (mis-)usage by generic users***

Some implementations do not allow at the moment to efficiently protect the spaces from mis-usage by generic users. As an example, it is possible that a generic VO user releases the space allocated to a VO. A request to read a file can trigger either a stage operation from tape in pools dedicated to production activities or internal copies between pools with no possibility to control such actions. These limitations apply especially to dCache and CASTOR installations. Some attempt has been made to protect the storage resources but a clean solution still is not available.

B. ***Full VOMS-awareness of the SRM implementations***

Some SRM implementations ~~such as dCache and CASTOR~~ are still not VOMS-FQAN aware at the authorization level. Even though authentication takes place at the SRM server using GSI, VOMS-FQANs are not used for authorization. This creates problems in managing the resources allocated to a VO and forces the use of specific proxies to execute certain tasks.

C. ***Selecting spaces for read operations in srmPrepareToGet, srmBringOnline, and srmCopy requests.***

The SRM v2.2 WLCG MoU specifies that clients must not specify a token for read operations. However, the document assumes silently that the file will be retrieved somehow in a "right" place. The SRM servers at the moment implements different behaviors: dCache serves the file from the place it was put, interfering with production operations; CASTOR honors the token if passed to select the pool where the file should be read from. This makes difficult the task to manage the space and to control the access. The possibility to specify a token also for read operation should be given.

D. ***Correct implementation of srmGetSpaceMetaData***

srmGetSpaceMetaData should return information about the space used and available (as defined in [4]) for a given token description and possibly for the so-called "default" pool. The default pool is used by those experiments who have decided not to use space tokens in their operations. Should srmGetSpaceMetaData be used to retrieve the needed information on space usage on the "default pool" or should another SRM method be introduced?

E. ***Providing the necessary information so that data could be efficiently stored on tapes.***

It is absolutely important to efficiently store data on tape sets so that data can be retrieved efficiently and minimizing the number of tape mounts to be executed. The SRM interface and WLCG clients should allow applications to pass all necessary information to perform the operation efficiently. Tokens should be passed to the tape backend as well as directory paths and any other useful info.

3. Some background on security

In this section we give some background information on VOMS groups and roles, VOMS FQANs and VOMS FQAN based Access Control Lists. We would also like to point out to the document in [3] where recommendations for changes in gLite authorization are given and suggestions on how to implement security in Storage and Data Management.

3.1 VOMS Groups, Roles, and Access Control Lists

Every user is assigned a VOMS proxy when using the WLCG Grid. In the context of this document a simple grid proxy is equivalent to a VOMS proxy with the (single) VO being the only extra attribute (determined from a grid-mapfile). A proxy is first characterized by the Subject Distinguished Name (DN), and can have extensions that define the privileges of the user holding that proxy at a given moment. Please check [2] for details. Example DN:

(1) /DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=flavia/CN=388195/CN=Flavia Donno

To define the privileges of a user at a given moment, groups, subgroups, and roles can be defined. In particular, a user can belong to multiple groups and sub-groups and have a number of roles at a given time. Example of groups and roles:

(2) /dteam/Role=lcgadmin
(3) /dteam
(4) /dteam/cern

Once the user presents his proxy to a Grid service, this typically maps the groups, subgroups, and roles to one or more GIDs (Group IDs) and the user DN to one specific UID (User ID). The privileges of the user on the resources managed by the contacted Grid service are therefore defined by the privileges of the n-tuple (UID, GIDs) in the system. LCMAPS mapping examples:

(5) "/dteam/Role=lcgadmin" .dteamsgm
(6) "/dteam" .dteam
(7) "/dteam/cern" .dteamcern

An Access Control List (ACL) is a list of entries defining the authorization on a given resource. ACLs can be positive, i.e. defining who is authorized to perform a given set of operations or access a given resource, or negative, negating permission to the service. Example of a DPM positive ACL:

(8)
```
# file:      /grid/dteam
# owner:  root
# group:   dteam
user::rwx
group:: rwx
group:dteam/Role=lcgadmin:rwx
group:dteam/Role=production:rwx
mask::rwx
other::r-x
default:user::rwx
default:group:rwx
```

default:group:dteam/Role=lcgadmin:rwx
default:group:dteam/Role=production:rwx
default:mask::rwx
default:other::r-x

4. WLCG proposed extensions

We specify here the properties of the SRM data objects that are being considered for addition to the SRM protocol.

4.1 Extensions of space properties

An SRM space is further characterized by the following properties:

- Group (***NEW***)
- Owner/Group Permission (Release, Update, Read-from-Space, Write-to-Space, Stage-to-Space, Remove-from-Space, Modify-ACL) (***NEW***)

A space which does not have an associated token description is known as ***DEFAULT*** space. There could be many default spaces in a system, each with different characteristics defined by the "properties" of each space and by the space permissions (see later).

A space belongs to an owner/group which is the initial creator of the space. A space can also be created by a site administrator statically. The site administrator defines the owner and main group for the space. A space owner has administrative privileges over the space. For example it can override/fix permissions and delete unwanted files. However it would not be bound to any VOMS FQAN, but to the definition of owner (DN) and possibly group (VOMS FQAN).

Permissions on the spaces are expressed in terms of DNs and/or VOMS FQANs [2].

The owner of a space can initially set ACLs on the space. This feature is not needed in WLCG.

In WLCG the storage resource administrator can set default ACLs on spaces via an administrative interface. Default ACLs are ACLs which are applied to the newly dynamically or statically reserved spaces. Furthermore specific space ACLs can be initially set by the storage resource administrator on a space.

ACLs define the set of operations that a user whose FQAN matches the space ACLs can perform on the space.

The set of possible operations are:

- ***Release*** : specifies which users/groups can release the space. This operation is not supported in WLCG since only the owner of the space or the administrator of the storage service can release spaces.
- ***Update*** : specifies which users/groups can update the space, such as modifying its size, lifetime, etc. – Is this really needed ?

- ***Read-from-Space*** : specifies which users/groups can perform srmPrepareToGet operations on this space.
- ***Write-to-Space*** : specifies which users/groups can perform srmPrepareToPut or srmCopy operations to this space.
- ***Stage-to-Space*** : specifies which users/groups can perform srmBringOnline operations to this space.
- ***Remove-from-Space*** : specifies which users/groups can perform srmPurgeFromSpace operations from this space.
- ***Modify-ACLs*** : specifies which users/groups can modify the ACLs on the space. This operation is not yet supported in WLCG since only the storage resource administrator can perform this operation.

Only the primary FQAN should be considered when matching ACLs.

ACLs can be positive or negative. Positive ACLs are meant to grant access while negative ACLs specify who should be negated the authorization to the resources the ACLs apply to. Negative ACLs take precedence over positive entries.

The semantic of positive and negative ACLs must be agreed. A proposal is the following:

A service should first scan the list of negative ACL and if there is a match with the current client (DN and first FQAN should be considered) then the access is denied. Then the service should scan the positive ACL and if there is a match with the current client (DN and first FQAN should be considered) then the access is granted. If there was no match, then the access is denied.

The owner user and group has an implicit entry on the positive list, meaning that if both the negative and positive ACLs are empty, then the access is granted for them. However if they are explicitly denied by the negative list, then they cannot access the space.

Management tools must be available to the resource administrator to manipulate ACLs directly and change the internal resolution of proxies. The set of allowed management operations must be agreed on.

4.2 SRM v2.3 spaces management calls

In what follows we list the new SRM space management functions that would offer an interface to the requested new functionality.

4.2.1 srmReserveSpace

This function is used to reserve a space in advance for the upcoming requests to get some guarantee on the file management. Asynchronous space reservation may be necessary for some SRMs to serve many concurrent requests.

Please note: The proposed change implies modification of the current SRM v2.2 WSDL. We do not propose to change the WSDL at the moment. However we list the needed modification for supporting dynamic space reservation in the future with the requested permission on spaces.

Additional Data Types

```
Enum        TSpaceRequestType {
                                RELEASE,
                                UPDATE,
                                READ-FROM-SPACE,
                                WRITE-TO-SPACE,
                                STAGE-TO-SPACE,
                                REMOVE-FROM-SPACE,
                                MODIFY-ACLs}

typedef     struct {
                    string    FQAN,
                    TSpaceRequestType       spaceRequest
            } TAccessControlList
```

Parameters

In:		
string	authorizationID,	
string	userSpaceTokenDescription,	
string	Owner (VOMS DN),	
string	Group (VOMS FQAN),	
TAccessControlList[]	positiveACLs,	
TAccessControlList[]	negativeACLs,	
TRetentionPolicyInfo	retentionPolicyInfo,	
unsigned long	desiredSizeOfTotalSpace,	
unsigned long	desiredSizeOfGuaranteedSpace,	
int	desiredLifetimeOfReservedSpace,	
unsigned long []	arrayOfExpectedFileSizes,	
TExtraInfo[]	storageSystemInfo,	
TTransferParameters	transferParameters	
Out:		
TReturnStatus	returnStatus,	
string	requestToken,	
int	estimatedProcessingTime,	
TRetentionPolicyInfo	retentionPolicyInfo,	
unsigned long	sizeOfTotalReservedSpace,	// best effort
unsigned long	sizeOfGuaranteedReservedSpace,	
int	lifetimeOfReservedSpace,	
string	spaceToken	

4.2.2 srmGetSpaceMetadata

This function is used to get information of a space. Space token must be provided, and space tokens are returned upon a completion of a space reservation through *srmReserveSpace* or *srmStatusOfReserveSpaceRequest*.

Please note: The proposed change implies modification of the current SRM v2.2 WSDL. We do not propose to change the WSDL at the moment. However we list the needed modification for supporting dynamic space reservation in the future with the requested permission on spaces.

Additional Data Types

```
typedef     struct {
                string                  spaceToken,
                TReturnStatus           status,
                TRetentionPolicyInfo    retentionPolicyInfo,
                TAccessControlList[]    positiveACLs,
                TAccessControlList[]    negativeACLs,
                string                  owner,
                string                  group
                unsigned long           totalSize,          // best effort
                unsigned long           guaranteedSize,
                unsigned long           unusedSize,
                int                     lifetimeAssigned,
                int                     lifetimeLeft
} TMetaDataSpace
```

Parameters

```
In:
string          authorizationID,
string[]        arrayOfSpaceTokens

Out:
TReturnStatus   returnStatus,
TMetaDataSpace[]    arrayOfSpaceDetails
```

4.3 SRM v2.3 get calls

The definition of the srmPrepareToGet/srmStatusOfGetRequest, srmBringOnline/srmStatusOfBringOnline, and srmCopy/srmStatusOfCopy request remains unchanged with respect to what has been specified in the SRM v2.2 specification [1]. The only difference is that now clients can pass a storage token that has to be honored in the calls. A copy of the file associated to the list of SURLs specified must be retrieved in the space specified if the user making the request has the privileges to recall a copy of that file in the specified space.

4.4 Tape usage optimization

All relative SRM calls for tape usage optimization have the extra parameter:
TExtraInfo[] defined as follows:

typedef struct { string _key,_
string value
} **TExtraInfo**

TExtraInfo is used wherever additional information is needed. For example, when it is used for additional information for transfer protocols, the keys may specify access speed, available number of parallelism, and other transfer protocol properties. Servers must honor in all relative calls such parameter when passed. Furthermore, information about directory paths and tokens should also be kept into consideration when selecting tape sets.

REFERENCES

[1] The Storage Resource Manager Interface Specification, Version 2.2, OGF – Grid Storage Resource Management Working Group, 15 February 2008, http://datagrid.lbl.gov/tmp2/OGF-GSM-SRM-v2.2-080121.doc

[2] A VOMS Attribute Certificate Profile for Authorization, V. Ciaschini, 15 April 2004, http://grid-auth.infn.it/docs/AC-RFC.pdf

[3] Recommendations for changes in gLite Authorization, C. Witzig, 6 February 2008, https://edms.cern.ch/document/887174/1

[4] Storage Element Model for SRM 2.2 and GLUE schema description, F. Donno et al., v. 3.5, 27 October 2006, https://forge.gridforum.org/sf/docman/do/downloadDocument/projects.glue-wg/docman.root.background.specifications/doc14619;jsessionid=58E33DC10A69FABED90ACD4C8EFE6E1F