ISGC 2007
International Symposium on Grid Computing
Taipei, March 26 – 29, 2007

A Model for the Storage Resource Manager

Andrea Domenici
DIIEIT University of Pisa and INFN
Andrea.Domenici@iet.unipi.it

Flavia Donno
CERN, European Organization for Nuclear Research
Flavia.Donno@cern.ch

Outline

- Storage Elements
- The Storage Resource Manager
- Modeling the SRM
- Spaces, files, and their properties
- The static model
- The dynamic model
- A more formal static model
- Validation of existing SRM implementations
- Conclusions

Introduction

The HEP community at CERN bases its research on the data acquired by the Large Hadron Collider (LHC) to explore the fundamental laws of the Universe. Several Petabytes (10-15) of data will be collected by the 4 experiment detectors every year.

The Worldwide LHC Computing Grid (WLCG) is one of the largest Grid infrastructures serving the HEP community. The WLCG counts today more than 200 computing sites all over the World.

Because of its mission, one of the critical issues that WLCG has to face is the provision of a Grid storage service that allows for dynamic space allocation, the negotiation of file access protocols, support for quality of storage, authentication and authorization mechanisms, storage and file management, scheduling of space and file operations, support for temporary files, etc.

Storage Elements

A *Storage Element* (SE) is a Grid Service that provides:

- A mass storage system (MSS) that can be provided by either a pool of disk servers or more specialized high-performing disk-based hardware, or disk cache front-end backed by a tape system.
- A storage interface to provide a common way to access the specific MSS, no matter what the implementation of the MSS is.
- A GridFTP service to provide data transfer in and out of the SE to and from the Grid.
- Local POSIX-like input/output calls providing application access to the data on the SE.
- Authentication, authorization and audit/accounting facilities.

The Storage Resource Manager

The *Storage Resource Manager* (SRM) is a middleware component whose function is to provide dynamic space allocation and file management on shared storage components on the Grid.

More precisely, the SRM is a Grid service with several different implementations. Its main specification documents are:

- A. Sim, A. Shoshani (eds.), The Storage Resource Manager Interface Specification, v. 2.2, available at `http://sdm.lbl.gov/srm-wg/doc/SRM.v2.2.pdf`.
- P. Badino et al., Storage Element Model for SRM 2.2 and GLUE schema description, v3.5.

The Storage Resource Manager interface

The SRM Interface Specification lists the service requests, along with the data types for their arguments.

Function signatures are given in an implementation-independent language and grouped by functionality:

- *Space management* functions allow the client to reserve, release, and manage spaces, their types and lifetimes.
- *Data transfer* functions have the purpose of getting files into SRM spaces either from the client's space or from other remote storage systems on the Grid, and to retrieve them.
- Other function classes are *Directory*, *Permission*, and *Discovery* functions.

Some space management functions

srmReserveSpace allows the requester to allocate space with specified properties.

srmReleaseSpace releases an occupied space. If the space contains copies of a file, the system must check if those copies can be deleted.

srmChangeSpaceForFiles is used to change the space where the files are stored.

srmExtendFileLifeTimeInSpace is used to extend the lifetime of files that have a copy in the space.

Some data transfer functions

srmPrepareToPut creates a handle that clients can use to create new files in a storage space or overwrite existing ones.

srmPutDone tells the SRM that the write operations are done.

srmCopy creates a file by copying it in the SRM space.

srmBringOnline is used to make files ready for future use. The system may stage copies from tape to disk.

srmPrepareToGet returns a handle to an online copy of the requested file.

srmReleaseFiles marks as releasable the copies generated by srmPrepareToGet or srmBringOnline.

srmAbortRequest, srmAbortFiles force termination of asynchronous requests.

srmExtendFileLifeTime extends the (pin) lifetime of files, copies, or handles.

Modeling the SRM

The Interface Specification has the purpose of defining the SRM API, therefore it is not meant to provide an overall view of the underlying concepts.

The GLUE schema is a UML model meant to define only the SRM properties relevant for the Information Service, so it cannot fully represent the SRM and particularly its behavior.

A full-fledged model should complement the Interface Specification and the GLUE schema, and be:

- clear;
- precise;
- useful for all people involved.

Spaces, files, and their properties

A *space* is a portion of storage allocated to a user or a VO. Its main properties are:

- **Retention policy**, likelyhood of file loss: REPLICA, OUTPUT, CUSTODIAL.
- **Access latency**, readiness of file access: ONLINE (e.g., disk), NEARLINE (e.g., tape).

A *storage class* is a combination of retention policy and access latency. A file has a required storage class and a storage type related to the file's lifecycle:

- **Volatile**: limited lifetime, file is deleted by the SRM after expiration.
- **Durable**: limited lifetime, file must be deleted by the owner after expiration.
- **Permanent**: unlimited lifetime, file may be deleted by the owner.

SURLs and TURLs

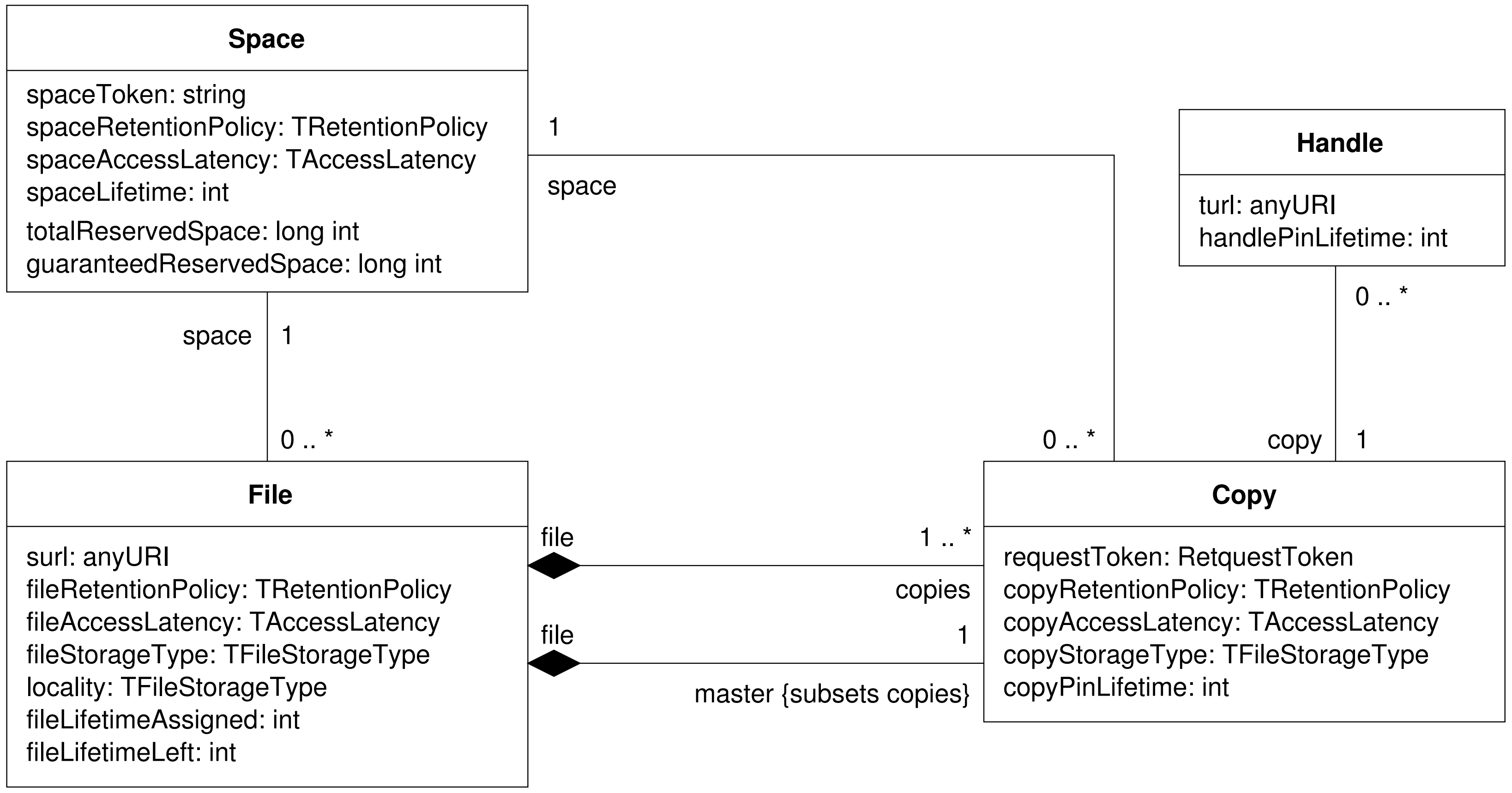
SURL (Storage URL) identifies a file in the logical namespace of a storage system. For example:

```
srm://dcache.fnal.gov:8443/somepath/vopath/filename
```

TURL (Transport URL) identifies an accessible copy in a storage system and includes a transfer protocol that can be used to access it. For example:

```
gsiftp://dcache.fnal.gov:2118/someinternalpath/filenam
```

The static model (1)

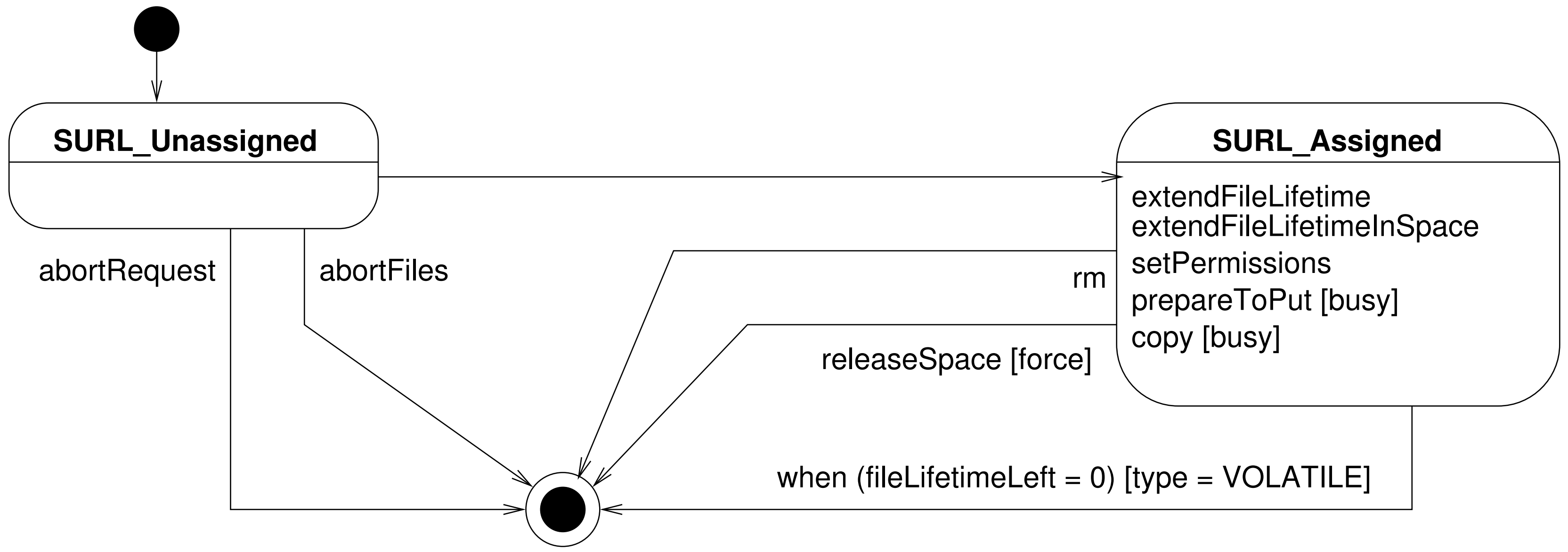

Space
spaceToken: string
spaceRetentionPolicy: TRetentionPolicy
spaceAccessLatency: TAccessLatency
spaceLifetime: int
totalReservedSpace: long int
guaranteedReservedSpace: long int
Handle
turl: anyURI
handlePinLifetime: int
File
surl: anyURI
fileRetentionPolicy: TRetentionPolicy
fileAccessLatency: TAccessLatency
fileStorageType: TFileStorageType
locality: TFileStorageType
fileLifetimeAssigned: int
fileLifetimeLeft: int
Copy
requestToken: RetquestToken
copyRetentionPolicy: TRetentionPolicy
copyAccessLatency: TAccessLatency
copyStorageType: TFileStorageType
copyPinLifetime: int
1
space
space
1
0 .. *
0 .. *
0 .. *
copy
1
file
1 .. *
copies
file
1
master {subsets copies}

The static model (2)

- A File has one or more Copies.
- A File has one master Copy.
- A Space holds zero or more Copies (possibly of different files).
- A file resides in the space holding the file’s master copy.
- A Copy is referred to by zero or more Handles.

The dynamic model (1)

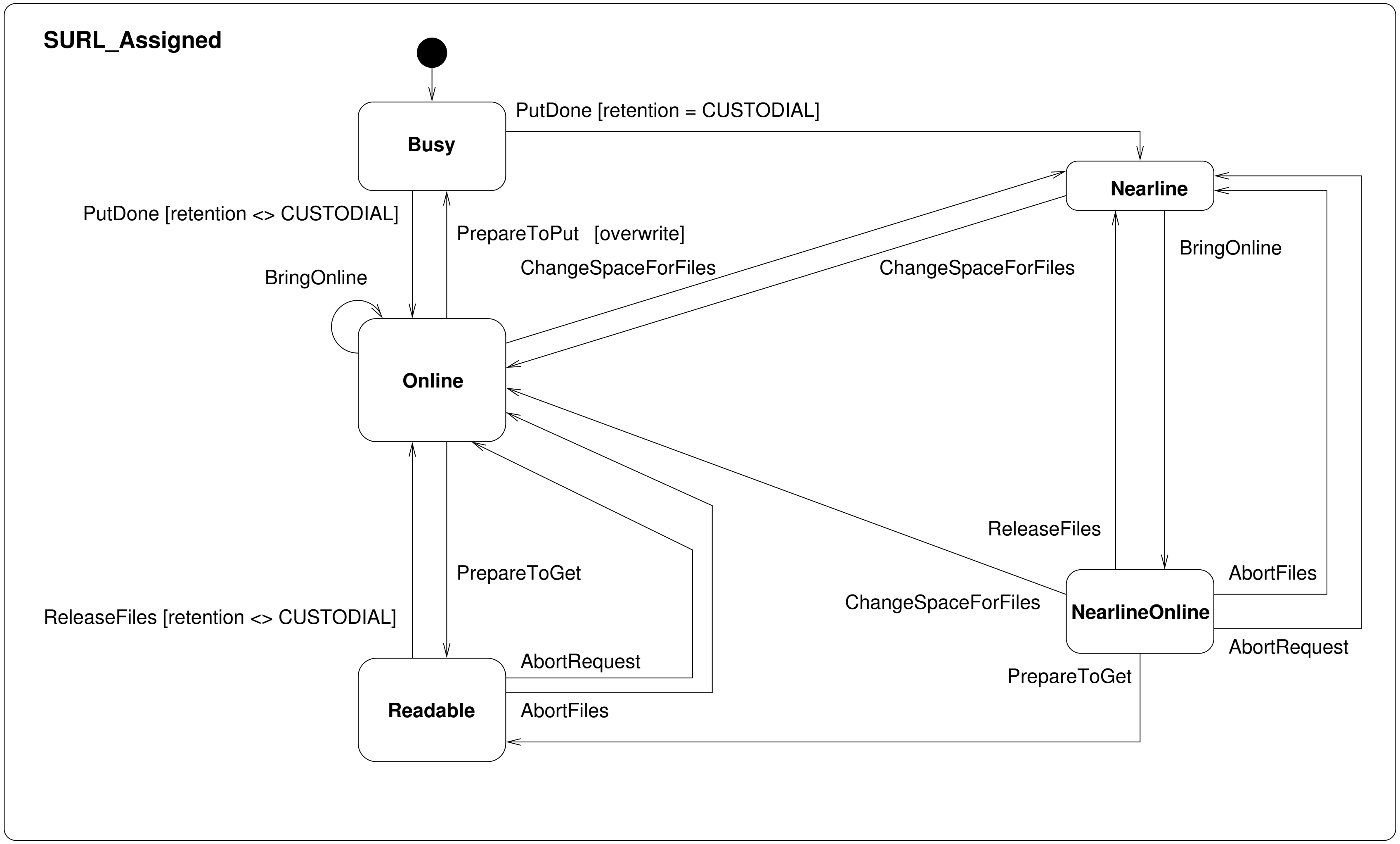
Top-level states of a File



- A File is created by **prepareToPut** or **copy**.
- **SURL_Unassigned** is a waiting state before a SURL is assigned.
- In state **SURL_Assigned** we list the request that leave the state unchanged.
- Other requests lead to the destruction of the file.

The dynamic model (2)

Substates of SURL_Assigned



The dynamic model (3)

- In state **Busy**, a handle is available to write data to (disk) storage.
- When the data have been written, the file becomes **Online** or **Nearline** according to its retention policy.
- In state **Readable**, a handle is available to read data from (disk) storage.
- In state **Nearline**, all copies are on a nearline space (tape).
- In state **NearlineOnline**, a copy is also on an online space (disk).

A more formal static model

The model is defined in terms of:

- **Basic sets** of discrete values for identifiers or properties.
- **Constructed sets**, Cartesian products of simpler sets.
- **Functions**, relating elements of the model.
- **Constraints**, statements about the model elements.

For constructed sets we show their *characteristic tuple*, showing the structure of a generic set element, e.g.:

$$\text{Storage class} \quad Sc = Rp \times Al$$

$$\langle retpol, latency \rangle$$

i.e., an element of Sc has two components, $retpol \in Rp$ and $latency \in Al$. The value of $retpol$ for an element p is $p.retpol$.

Common properties

Sizes	Sz	=	$\mathbb{N}$
Lifetimes	L	=	$\mathbb{N} \cup \{\top\}, \forall_{t \in L}\, t \leq \top$
Retention policy	Rp	=	$\{\text{REPLICA}, \text{OUTPUT}, \text{CUSTODIAL}\}$
Access latency	Al	=	$\{\text{ONLINE}, \text{NEARLINE}\}$

$$\text{REPLICA} < \text{OUTPUT} < \text{CUSTODIAL}$$

$$\text{ONLINE} < \text{NEARLINE}$$

Storage class $Sc = Rp \times Al$

$$\langle retpol, latency \rangle$$

Space

Protocols	P	=	$\{$rfio, dcap, gsiftp, file$\}$
Access Pattern	Ap	=	$\{$TRANSFER, PROCESSING$\}$
Connection Type	Ct	=	$\{$WAN, LAN$\}$
Space Tokens	T		a countable set of symbols
Space requests	R_s		a finite set of symbols

Properties $Prop = Sc \times P \times Ap \times Ct$

$$\langle sclass, protocol, access, connection \rangle$$

Spaces $S = T \times L \times Prop \times Sz \times R_s$

$$\langle token, lifetime, props, size, request \rangle$$

Copy and handle

Physical File Names	Pfn	a countable set of symbols
Copy requests	R_c	a countable set of symbols

Copies $C = Pfn \times L \times R_c$

$$\langle physname, pintime, request \rangle$$

TURLs	Tr	a countable set of symbols
Handle requests	R_h	a countable set of symbols

Handles $H = Tr \times L \times R_h$

$$\langle turl, pintime, request \rangle$$

File

SURLs	Sr	a countable set of symbols
File Types	Ft	{FILE, DIR}
Creation Time	T_c	$\mathbb{N}$
Storage Types	St	{VOLATILE, DURABLE, PERMANENT}
File Locality	Fl	{ONLINE, ONLINE_NEARLINE, NEARLINE, UNAVAILABLE, LOST}

Files $F = Sr \times Ft \times Sz \times T_c \times St \times Sc \times Fl$

$$\langle surl, ftype, size, ctime, stype, sclass, locality\rangle$$

Functions

- Start time of a space, copy, or handle:

$$stime : S \cup C \cup H \rightarrow \mathbb{N}$$

- Remaining (pin) lifetime at time t:

$$lleft : (S \cup C \cup H) \times \mathbb{N} \rightarrow \mathbb{N}$$

- Set of files resident on a space:

$$resfiles : S \rightarrow \mathcal{P}F$$

- The space holding a file's master copy:

$$mspace : F \rightarrow S$$

Constraints on files

A file must reside in one space:

$$\forall_{f \in F} \exists_{1 s \in S}\ f \in resfiles(s)$$

File retention policy must match space retention policy:

$$\forall_{f \in F}\ f.sclass.retpol = mspace(f).props.sclass.retpol$$

Space latency must satisfy file latency requirement:

$$\forall_{f \in F}\ f.sclass.latency \geq mspace(f).props.sclass.latency$$

A file cannot outlive its space:

$$\forall_{f \in F, t > stime(f)}\ 0 < lleft(f, t) < lleft(mspace(f), t)$$

Validation of existing SRM implementations

When defining a protocol, it is very important to validate it on specific implementations.

In particular, since SRM focuses on providing an interface to Mass Storage Systems, it was extremely critical to test its implementations on several MSS back-ends.

Therefore a testbed with 5 MSS systems supporting SRM 2.2 has been established.

The SRM testbed

CASTOR developed at CERN and used by many other labs to serve data on automatic tape libraries and on disk servers used mainly as a front-end cache. The SRM 2.2 implementation for CASTOR has been made by RAL (UK).

dCache developed at DESY (Germany), used by many sites with multiple MSS backends, both custom and proprietary. dCache can be used also as a disk-only MSS. The SRM 2.2 implementation for dCache has been made by FNAL.

DPM developed at CERN. This is a disk-only based MSS. The SRM 2.2 implementation has been made at CERN.

DRM/BeStMan is the LBNL disk-based storage system. LBNL has been the first promoter of SRM. This storage system was the first prototype on which SRM has been tested.

StoRM is a disk-based system. It offers an SRM 2.2 interface to parallel file systems such as GPFS or PVFS. The SRM 2.2 implementation has been made at CNAF.

Test case families

Using various techniques of black-box testing, 5 families of test cases have been designed:

Availability to check the availability in time of the SRM service end-points.

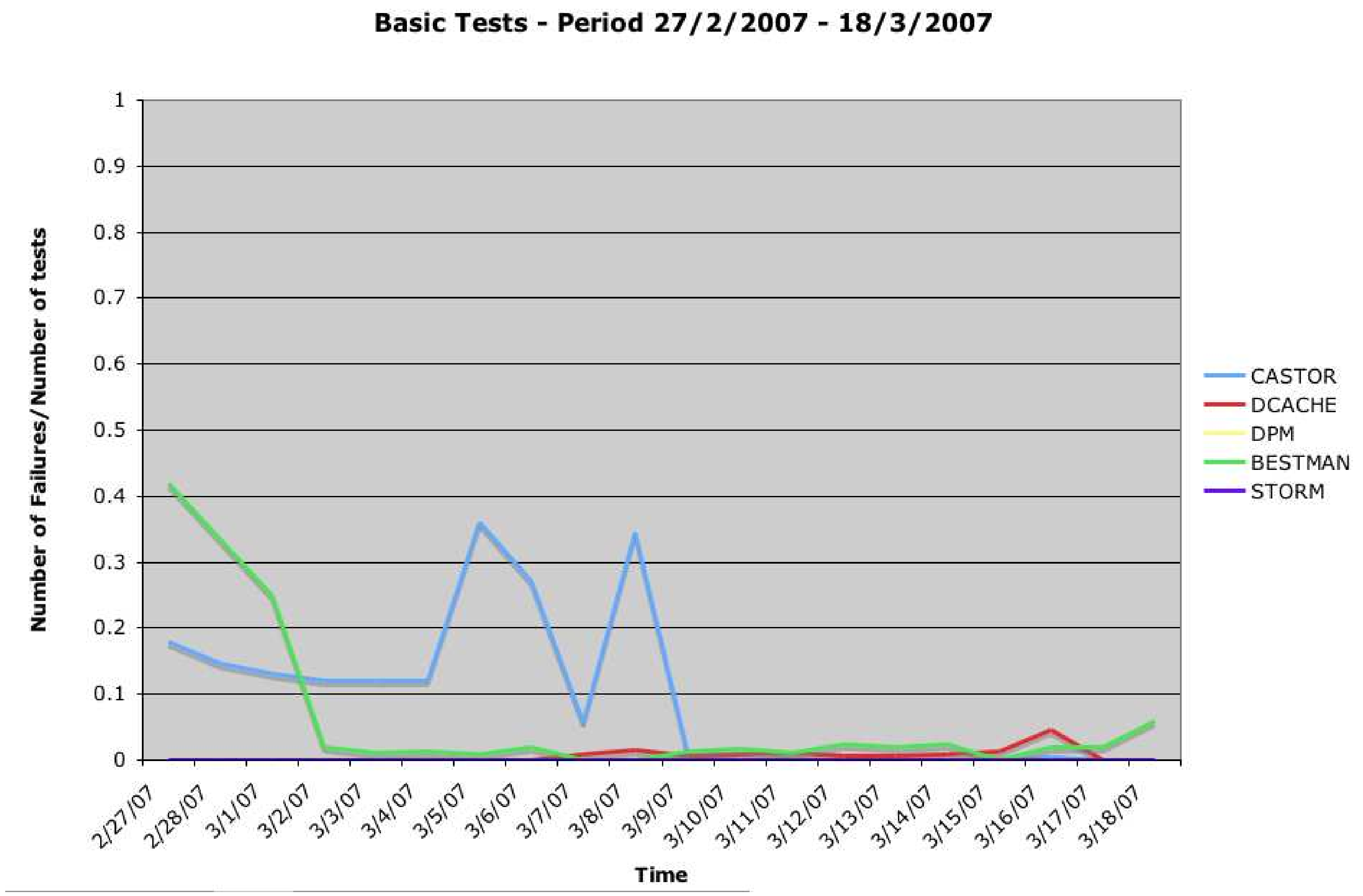
Basic to verify basic functionality of the implemented SRM APIs.

Use Cases to check boundary conditions, use cases derived by real usage, function interactions, exceptions, etc.

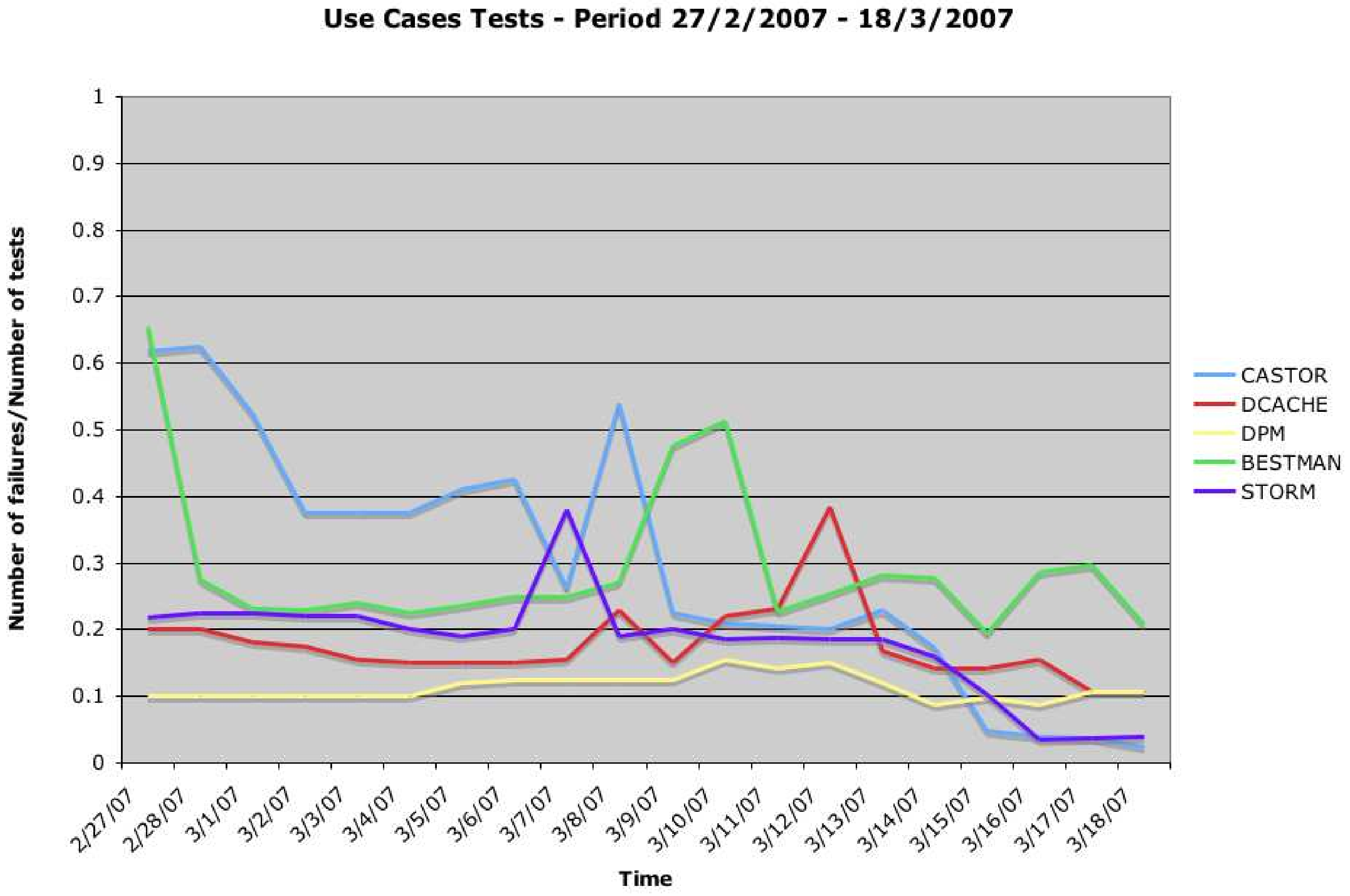
Exhaustion to exhaust all possible values of input and output arguments such as length of filenames, SURL format, optional arguments, strings, etc.

Stress tests to stress the systems, identify race conditions, study the behavior of the system when critical concurrent operations are performed, etc.

Summary of Basic test suite



Summary of Use Cases test suite



Conclusions

- A comprehensive model of the SRM is being developed to support the development and verification of SRM implementations.
- The first draft of the model is available, and feedback from its users is awaited.
- Developing the model has helped in identifying unanticipated behaviors and interactions.
- The analysis of the model has helped design a few families of tests for the validation of the protocol.
- The testing campaign itself has motivated the developers to reconsider many of the initial assumptions and decisions, leading to solutions that seem to better satisfy the needs of the users.
- Both testing and specification are still ongoing. WLCG is expected to include the SRM interface in its production environment most probably by July 2007.

Thank you

谢谢