A Model for the Storage Resource Manager

Andrea Domenici[1] and Flavia Donno[2]

[1] DIIEIT, University of Pisa, v. Diotisalvi 2, I-56122 Pisa, Italy
[2] CERN, European Organization for Nuclear Research, CH-1211 Geneva 23, Switzerland

Abstract. The Storage Resource Manager has been proposed as a standard interface for high-end storage systems deployed on Grid architectures. In this paper we propose a conceptual model for the SRM that should supplement its API specification with a clear and concise definition of its underlying structural and behavioral concepts. This model would make it easier to define its semantics, it would help service and application developers, and provide for a more rigorous validation of implementations. Different notations are used as appropriate to define different aspects of the model.

1 Introduction

The Worldwide LHC Computing Grid (WLCG) is the infrastructure that will provide the computational and storage facilities needed to process the data collected by the four experiments at the Large Hadron Collider (LHC) at CERN, amounting to several Petabytes each year.

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, and other storage management functions.

The *Storage Resource Manager* (SRM) has been proposed [1] as a standard interface for high-end storage systems deployed on Grid infrastructures. In particular, a significant effort by an international collaboration coordinated by the WLCG has led to the definition of the SRM 2.2 protocol and to its implementation on all the storage solutions deployed in WLCG.

The SRM is specified primarily as an application programming interface (API). In this paper we propose a conceptual model for the SRM that should supplement the API and other specifications with an explicit, clear and concise definition of its underlying structural and behavioral concepts. This model would make it easier to define the service semantics, it would help service and application developers and provide for a more rigorous validation of implementations.

The proposed model is meant to strike a satisfactory compromise between clarity and formality. Different notations (e.g., basic set-theoretic and logical formalism, UML [2] diagrams, and plain English) are used as appropriate to define different aspects of the model.

2 The Storage Resource Manager

A *Storage Element* (SE) is a Grid Service implemented on a mass storage system (MSS) that may be based on a pool of disk servers, on more specialized high-performing disk-based hardware, or on a disk cache front-end backed by a tape system, or some other reliable, long-term storage medium. Remote data access is provided by a GridFTP service [3] and possibly by other data transfer services, while local access is provided by POSIX-like input/output calls.

A SE provides *spaces* where users create and access *files*. A file is a logical set of data that is embodied in one or more physical *copies*.

Storage spaces may be of different qualities, related to reliability and accessibility, and support different data transfer protocols. Different users may have different requirements on space quality and access protocol, therefore, besides the data transfer and file access functions, a SE must support more advanced resource management services, including dynamic space allocation.

The *Storage Resource Manager* (SRM) is a middleware component that provides the resource management services through a standard interface, independent of the underlying MSS. The interface is defined by the *SRM Interface Specification* (IS) [4] that lists the service requests that a client application may issue, along with the data types for their arguments and results.

Request signatures are grouped by functionality, such as *space management* requests that allow clients to reserve, release, and manage spaces, specifying or negotiating their quality and lifetime, and *data transfer* requests that get files into SRM spaces either from a client's space or from other storage systems, and retrieve them. Other groups are *directory*, *permission*, and *discovery* functions.

3 A Model for the Storage Resource Manager

The main SRM specification are the above mentioned IS and the *Storage Element Model for SRM 2.2 and GLUE schema description* [5]. Other relevant documents are [1, 6, 7]. We proposed a model to extend the specifications with a synthetic description of the basic entities, their relationships, and their behaviors.

We have chosen to use two submodels, with different levels of formality. The semi-formal model uses plain English and UML diagrams, and it is meant to give an overall view of the system, identifying its main components, their relationships and behavior, and to define and clarify the terms used in the IS. A more formal model uses set-theoretic and logical notations to express constraints. This model is meant to resolve ambiguities that might remain in the semi-formal model, and to support the design and testing of SRM implementations.

3.1 Describing Concepts and Properties

In the *static model*, the SRM concepts are represented as object classes, their properties and reciprocal relationshisps being modeled by attributes and associations subject to various constraints. Figure 1 shows a partial UML class diagram for the SRM static model.

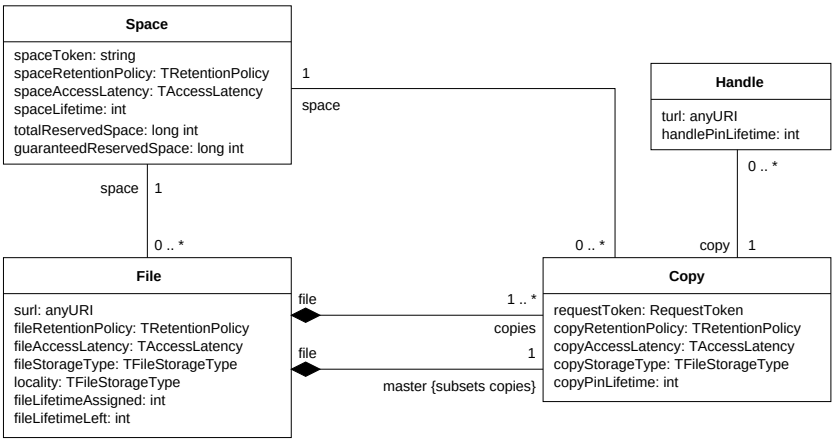

Fig. 1. Static model of the SRM.

Some of the attributes represent important properties: *retention policy*, *access latency*, and *(file) storage type*. Retention policy describes the likelihood that a file copy may be lost in a given storage space. This likelihood may be high (REPLICA retention policy), intermediate (OUTPUT), or low (CUSTODIAL). The OUTPUT policy is currently not implemented. Access latency describes data accessibility: A space where data are immediately accessible is ONLINE, otherwise it is NEARLINE. A NEARLINE space is supported by mechanical media, such as tape, that require data to be staged to temporary disk storage for access. A third latency, OFFLINE, is currently not implemented. The *storage type* refers to file lifetime. A VOLATILE file is deleted automatically after a given time. A DURABLE file also has a limited lifetime, but it must be removed explicitly by its owner. A PERMANENT file has an unlimited lifetime, until removed by its owner. Durable files are currently not implemented. A *Site URL* (SURL) identifies the file within the SE, and the SE itself.

The associations in the diagram show that a space hosts zero or more files, that each file has one or more copies, one of which is the master copy, that each copy resides on a space, and it has one or more handles.

3.2 Describing Behavior

The *dynamic model* of the SRM is described by UML state diagrams. Figure 2 shows a part of the model, related to files.

A file is created with a *prepareToPut* or a *copy* request. A request can be served asynchronously, so a file may remain for some time in a waiting state (*SURL_Unassigned*) before it is assigned a SURL. In this state, the file can be destroyed by an *abortFiles* or *abortRequest* operation. Otherwise, eventually a SURL is assigned and the file enters a state (*SURL_Assigned*) where it can be

destroyed by an *rm* (remove) request, by a *releaseSpace* request with the *force* option, when its lifetime expires and the file is volatile, or when the pin lifetime of its last copy expires and the file is volatile.

Some requests are accepted in the *SURL_Assigned* state, but they do not alter the behavior. Such requests are listed as *internal transitions* (shown inside the state icon in the diagram) and leave the file in *SURL_Assigned* and in its current substate, whichever it be.

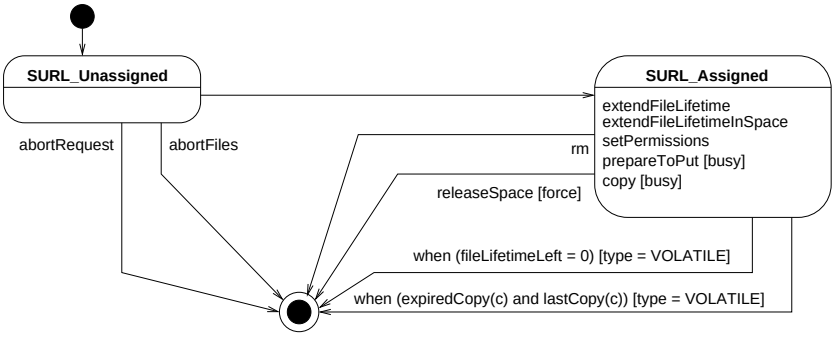

Fig. 2. State machine for File (1).

4 A More Formal Static Model

While the semi-formal model exposed above is helpful for users and developers of the SRM, a finer level of detail and a greater degree of formality are needed to ensure interoperability and full compliance with the specification. Therefore we propose an initial, still incomplete, formal model expressed in basic mathematical notation. Since the SRM is still evolving and several issues are still open, the model is limited to fundamental features, upon which further extensions and refinements can be built.

An elementary mathematical notation was chosen instead of some more specialized language, such as the UML *Object Constraint Language* [8] or the *Z Specification Language* [9], but it should be easy to translate the notation adopted here to those formalisms, if needed.

First, we introduce some *basic sets* whose members are unstructured values, such as atomic symbols (meant to represent names of objects or discrete values for their attributes) or numbers. Then we define the *constructed sets* of *storage elements*, *spaces*, *copies*, *handles*, and *files* as Cartesian products of some previouly defined sets.

Functions are used to represent various properties and relationships. *Constraints* on properties and relationships are expressed as predicate logic formulas.

For example, two of the basic sets are the set of space or file sizes, defined as identical to the set of natural numbers ($Sz = \mathbb{N}$), and the set of retention policies, defined as a set of three values ($Rp = \{\mathsf{REPLICA}, \mathsf{OUTPUT}, \mathsf{CUSTODIAL}\}$).

The set of spaces is defined as $S = T \times L \times Prop \times Sz \times R_s$, where T is the set of space tokens (i.e., identifiers), L the set of lifetimes, $Prop$ a set of tuples of space properties, and R_s is the set of request issued for each space.

As an example of a constraint, the statement that "*a file cannot outlive its space*" is expressed as

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

where F is the set of files, t is a time value, $stime(f)$ is the time of file creation, $lleft(\cdot, t)$ is the file or space lifetime remaining at time t, and $mspace(f)$ is the space holding the master copy of file f.

5 Validation of Existing SRM Implementations

The SRM has been implemented for five different MSSs, namely CASTOR [10], developed at CERN and based on tape libraries and disk servers, dCache [7], developed at DESY, used with multiple MSS backends, DPM [11], a disk-only MSS developed at CERN, DRM/BeStMan, a disk-based system developed at LBNL, the first promoter of SRM, and StoRM [12], a disk-based system developed at CNAF, based on parallel file systems such as GPFS or PVFS.

All these systems are being tested for compliance with the SRM IS. Using various techniques of black-box testing [13], five families of test cases have been designed: *Availability* to check the availability of the SRM end-points; *Basic* to verify basic functionality of the implemented SRM APIs; *Use Cases* to check boundary conditions, function interactions, and exceptions; *Exhaustion* to exhaust all possible values of input and output arguments such as length of filenames, SURL format, and optional arguments; *Stress tests* to stress the systems, identify race conditions, study the behavior of the system when critical concurrent operations are performed, and in other exacting conditions.

The SRM model proposed in this paper has been used to derive several test cases in the *Basic* and *Use Cases* test suites.

6 Conclusions

A comprehensive model of the SRM is being developed to support the development and verification of SRM implementations, using different notations and levels of formality in order to satisfy the needs of different stakeholders in the SRM development.

The first draft of the model is available, and feedback from its users is awaited. In fact, the model has already contributed to the validation of existing implementations by assisting in the design of a few families of tests, and its development has helped in identifying unanticipated behaviors and interactions. The testing campaign itself has helped the developers to find solutions that better satisfy the needs of the users.

7 Acknowledgements

The work of the many people in the SRM collaboration is gratefully acknowledged. In particular we would like to thank Arie Shoshani, Alex Sim and Junmin Gu from LBNL, Jean-Philippe Baud, Paolo Badino, Maarten Litmaath from CERN, Timur Perelmutov from FNAL, Patrick Fuhrmann from DESY, Shaun De Witt from RAL, Ezio Corso from ICTP, Luca Magnoni and Riccardo Zappi from CNAF/INFN, for their valuable input in the specification definition process. Finally, we would like to thank the WLCG project for giving us the opportunity to collect the requirements and test the proposed protocol for real use-cases. The authors have been supported by CERN and INFN, respectively.

References

1. Shoshani, A., Sim, A., Gu, J.: Storage Resource Managers: Middleware Components for Grid Storage. In: Proceedings of the 9th IEEE Symposium on Mass Storage Systems (MSS '02). (2002)
2. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. 2nd edn. Addison-Wesley (2004)
3. Allcock, W., et al.: GridFTP protocol specification. Document, GGF GridFTP Working Group (September 2002)
4. : The Storage Resource Manager Interface Specification, Version 2.2. `http://sdm.lbl.gov/srm-wg/doc/SRM.v2.2.pdf` (December 2006) Storage Resource Manager Working Group.
5. Badino, P., et al.: Storage Element Model for SRM 2.2 and GLUE schema description, v3.5. Technical report, WLCG (Oct. 27, 2006)
6. Shoshani, A., et al.: Storage Resource Management: Concepts, Functionality, and Interface Specification. In: Future of Grid Data Environments: A Global Grid Forum (GGF) Data Area Workshop, Berlin, Germany (March 9–13, 2004)
7. Ernst, M., et al.: Managed data storage and data access services for data grids. In: Proceedings of the Computing in High Energy Physics (CHEP) conference, Interlaken, Switzerland (September 27 – October 1, 2004)
8. Warmer, J., Kleppe, A.: The Object Constraint Language: Getting Your Models Ready for MDA. 2nd edn. Addison-Wesley (2004)
9. Woodcock, J., Davies, J.: Using Z – Specification, Refinement, and Proof. Prentice Hall (1996)
10. Barring, O., et al.: Storage Resource Sharing with CASTOR. In: 12th NASA Goddard/21st IEEE Conference on Mass Storage Systems and Technologies (MSST2004), U. of Maryland, Adelphi, MD (Apr. 13–16, 2004)
11. Baud, J.P., Casey, J.: Evolution of LCG-2 Data Management. In: Proceedings of the Computing in High Energy Physics (CHEP'04) conference, Interlaken, Switzerland (September 27 – October 1, 2004)
12. Corso, E., et al.: StoRM, an SRM Implementation for LHC Analysis Farms. In: Proceedings of the Computing in High Energy Physics (CHEP'06) conference, Mumbai, India (Feb. 2006)
13. Myers, G.J., (rev. by), C.S., (rev. by), T.B., (rev. by), T.M.T.: The Art of Software Testing. 2nd edn. John Wiley & Sons (2004)