Project number: **RI 312579**

Project acronym: **ER-flow**

Project full title:
Building an European Research Community through Interoperable Workflows and Data

Theme: **Research Infrastructures**
Call Identifier: **FP7-Infrastructures-2012-1**

Funding Scheme: **Coordination and Support Action**

MS4.1: Data objects transfer service

CNRS

Due date of milestone: 01/09/2013 Start date of project: 01/09/2012	Actual submission date: 09/09/2013 Duration: 24 months
Dissemination Level: PU	

Contents

List of Figures

List of Tables

Status and Change History

Status	Name	Date	Signature
Draft	T. Glatard	5/9/13	n.n electronically
Reviewed	J. Montagnat	9/9/13	n.n electronically
Approved	K. Eigelis	13/9/13	n.n electronically

Table 1: Status of deliverable

Version	Date	Pages	Author	Modification
0.1	5/9/2013	all	T. Glatard	first draft
0.2	9/9/2013	all	T. Glatard and J. Montagnat	final document
0.3	17/9/2013	5,6,9,10	T. Glatard and J. Montagnat	Added context about storage backends in ER-flow (pages 5-6 and reference page 10). Mentioned globusonline.eu in conclusion (page 9).

Table 2: Change History

1 Introduction

Creating meta-workflows from workflows of different languages running on different Distributed Computing Infrastructures (DCI) has been proposed and developed in the SHIWA project, and is now being offered to user communities through the ER-flow Science-Gateway. However, designing fully functional meta-workflows is still clumsy because of the need to handle peculiarities of the underlying DCIs while designing meta-workflow. In particular, meta-workflows have to be significantly instrumented to allow data transfers between embedded (a.k.a *native*) workflow systems, as outlined in D4.1 study on data interoperability [2]. A corollary is the need for users to transfer input data to the native systems connected to the inputs of the meta-workflow, and to transfer results from the native systems connected to the output of the meta-workflow.

File transfers in Science-Gateways mainly occur between users hosts, job execution hosts, and DCI storage systems. User hosts denote machines were input data usually reside, typically personal workstations or file servers. Job execution hosts are DCI worker nodes, where jobs generated by workflows actually execute. DCI storage systems are the storage hosts from where computing jobs usually download their input data, and where they store output data. Files can be workflow inputs, temporary data transferred between native workflow systems, and workflow outputs.

This document proposes an architecture to facilitate the handling of data transfers inside meta-workflows, and between users and meta-workflows. Section 2 identify desirable features for a data transfer service, and section 3 presents a first architectural design.

2 Desirable features

This section lists the main desirable features of the system, which have been collected from our experience with the SHIWA platform and other Science-Gateways.

A uniform indexing space. First, a data transfer service must provide an indexing space where files are represented by uniform resource identifiers (URIs). File management systems used in DCI, such as file catalogs, already provide uniform indexing space, but they have to be harmonized when multiple systems are used. A unifying system could then build on top of the existing uniform indexing spaces, or re-aggregate files located on the various sites available to the DCIs.

A RESTful web API. This uniform file indexing space should preferably be accessible through a RESTful web API in order to enable access from a large set of existing clients such as `wget`, `curl` and others. The adoption of grid technologies has been limited by their initial use of new APIs to access storage, such as the LCG File Catalog[1] in EGI, the Globus Replica Location Service[2] and the Storage Resource Manager[3]. As explained in the introduction of this document, files handled by Science-Gateways are usually accessed by multiple users, jobs, workflow systems and DCIs. It is thus important that file management interface stay general, such as offered by RESTful APIs.

Multiple, heterogeneous storage backends. Data handled by Science-Gateways is usually stored on multiple storage systems accessible through heterogeneous interfaces and protocols, and sometimes federated by file catalogs. For instance, in ER-flow, the Astronomy community uses a distributed file catalogue called the Virtual Observatory[4], the MoSGriD community indexes files

[1] https://twiki.cern.ch/twiki/bin/view/LCG/LfcAdminGuide#LFC_CLI_and_API
[2] http://www.globus.org/rls/
[3] https://sdm.lbl.gov/srm
[4] http://www.ivoa.net

in a database accessible through the Molecular Simulation Grid Portal[5], and the Life-Science community in ER-flow uses EMI's LCG File Catalog (LFC). More details about ER-flow's community requirements are found in MS5.1 [1]. A combination of several file transfer and referencing technologies has to be handled.

Monitoring interfaces. File transfers from user workstations may take a long time to complete, due to the number of transferred files or their size. Therefore, transfers must be seen as tasks for which a monitoring interface should be offered. This interface should allow to monitor the status of the on-going transfers, the history of the past transfers, and the restart/recovery of failed transfers.

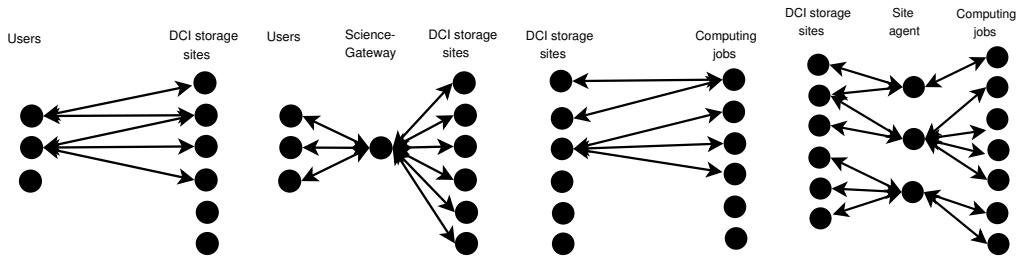
Asynchronous interfaces. Another feature coming from the frequent long duration of file transfers is the ability to perform them asynchronously. Transfers of large files or large collections of files should not block the user's host, and be robust to machine reboots or network interruptions. Trasnfers should be executed off-line, while end-user clients are disconnected from the service. A few services such as Dropbox[6] and Globus Online[7] now provide this functionality.

No instrumentation of native workflow engines. A design choice in SHIWA has been to instrument native workflow engines as little as possible so that their integration could be performed with minimal effort. While this puts more efforts on the meta-workflow engine, we believe that this principle should be kept to avoid any modification in the native engines.

Simple authentication. Managing personal credentials for multiple DCIs has been attempted several times with mixed results but additional burden for users and service developers. For this reason, we believe that a data transfer system should use its own, robot credentials to initiate file transfers. These could take the form of a robot certificate[8] or other forms of credentials allowing applications to access services, for instance OAuth 2.0[9]. Using robot credentials usually requires that personalized logs of operations are kept, in our case the association between ER-flow users and file transfers.

Performance, scalability, reliability. Performance (transfer time), scalability, and reliability are consequences of trade-offs between centralized and de-centralized architectures adopted for user↔Science-Gateway transfers and Science-Gateway↔jobs transfers. These different architectures are illustrated on Figure 1. Decentralized architectures improves scalability due to the absence of any single point of failure. For instance, decentralized user↔DCI was adopted by grid browsers such as VBrowser[10] and the Java Universal eXplorer (JUX). These tools allowed users to directly browser multiple grid resources from their workstation. Reliability, however, remains an issue with these systems since connectivity between multiple users and multiple DCI storage systems is difficult to maintain and debug. Consequently, most Science-Gateway adopted a centralized architecture to handle file transfers between DCIs and users. Performance and scalability issues related to centralized architectures can be mitigated by caches and asynchronous interfaces. Conversely, file transfers between DCIs and jobs remain mostly de-centralized, each individual job contacting storage resources to transfer their own files. While this approach is meant to improve scalability (the number of jobs handled by a Science-Gateway is usually greater than its number of users), it also has performance and reliability issues when the number of storage resources or jobs

[5] https://mosgrid.de
[6] http://dropbox.com
[7] http://www.globusonline.org
[8] https://wiki.egi.eu/wiki/EGI_robot_certificate_users
[9] http://oauth.net
[10] http://sourceforge.net/projects/vlet/files



(a) Users↔DCI, decentralized (b) Users↔DCI, centralized (c) Jobs↔DCI, decentralized (d) Jobs↔DCI, hierarchical

Figure 1: Architectures for file transfers in Science-Gateways

increases. File replication on multiple storage sites is a common strategy to cope with this. However, efficient automatic replication strategies are seldom used in production systems [4], which makes it clearly out of the scope of ER-flow. An interesting trade-off between centralized and de-centralized DCI↔job file transfers is to pre-stage files to computing sites before execution, resulting in a hierarchical architecture for file transfers (see Figure 1(d)). For instance, this approach is adopted in the ARC middleware[11]. Interesting transpositions of this hierarchical approach could be studied and implemented on different DCIs.

3 System design

Following the analysis presented above, we suggest in this section a system design to facilitate data transfers among workflow engines.

3.1 System components

Uniform indexing space. We propose to rely on the file naming systems used in the different DCIs to uniquely index files in ER-flow. In practice, it means that each storage system integrated in ER-flow should be checked to make sure that its file identifiers can be converted to URIs supported by ER-flow. ER-flow should maintain a list of supported protocols, and possibly conversion rules between URIs and DCI-specific file names. For instance, EGI logical file names stored in the *biomed* VO are sometimes written as a path such as `/grid/biomed/file.txt`. Such paths should be converted to URIs using contextual information about the *biomed* VO, resulting for instance in URI `lfn://lfc.biomed.in2p3.fr/grid/biomed/file.txt`. Files stored in the Science-Gateway itself must also be indexed. For instance, a file stored in a WS P-Grade portal and used in job input or output sandboxes could be indexed using the URL of the portal, and possibly user- or job-specific identifiers: `http://portal.somewhere/userA/job1/file.txt`.

API. A RESTful API must be defined to support the operations related to file transfers. We propose the following methods for this API (Java-like syntax):

1. `URI generateURI (String path, Enum system, String context)`: this method converts a path to an ER-flow URI. If `path` is already a URI with a protocol supported by ER-flow, it is returned as is. Otherwise, a URI is built using `system`-specific conversion rules parametrized by `context`. Valid values for `system` are provided by the enumeration of workflow engines supported by ER-flow.

[11] http://www.nordugrid.org/arc

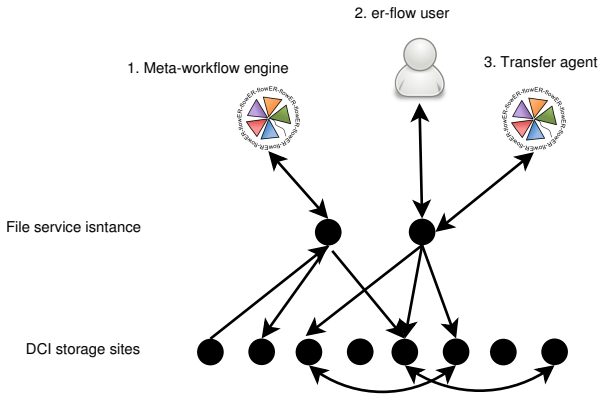


Figure 2: Use cases that can be implemented using the file service.

2. `URI transfer (URI source, URI destination)`: this method initiates a file transfer from the source URI to the destination URI, returning an identifier for this transfer. This identifier must be a URI allowing to locate the service in charge of this transfer. If source (resp. destination) is a local URI, e.g. `file:///tmp.file.txt`, then the transfer results in a file upload (resp. download) from the local machine. File transfers *may* be performed using this method, or using any other native DCI client. In case the DCI doesn't provide an ER-flow-supported URI, `generateURI` can be used.

3. `Status monitor (String transferID)`: this method returns the status of a file transfer.

Additional methods could also be added to control the file transfer life-cycle (for instance `cancel` or `suspend`/`resume`) or to get information about the system (e.g. what are the supported systems).

File service. The file service implements the methods of the API. For performance, scalability and reliability purposes, several file service instances can co-exist. File service instances do not store any persistent information about files, therefore different services can be used concurrently for most operations, even to access the same file. For instance, a file can be transferred from DCI X to DCI Y using a given file service instance, and be downloaded from DCI Y using another one. However, file service instances do store persistent information about transfers, which means that a given file transfer has to be handled by a single file service instance. The file service must be able to handle multiple storage backends. This can be achieved using existing solutions such as a service like SCI-BUS Data Bridge, or libraries like JSAGA.

3.2 Use cases

Figure 2 summarizes the use cases that can be implemented with the file service. As explained above, the number of file service instances can be increased to accommodate performance, scalability and reliability issues.

Data transfers in meta-workflows. The file service can be used by meta-workflow engines to simplify file exchanges between native workflows. Assuming that the enacted meta-workflow consists of a native workflow A producing a single file consumed by a native workflow B (see Figure 3), then the meta-workflow engine could implement the following sequence:

1. Generate a URI for file produced by native workflow A (calls method `genrateURI`).

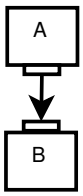


Figure 3: Simple meta-workflow where native workflow A produces a file consumed by native workflow B.

2. If URI cannot be handled by native workflow B, transfer URI to a storage system that native workflow B can handle (calls method `transfer`). Monitor transfer and send destination URI to native workflow B when transfer completes.

3. If native workflow B requires file content, transfer URI locally (calls method `transfer`) and pass it to native workflow B.

Information about what storage system is supported by a given native workflow engine is obviously required for the meta-workflow engine to implement this scenario. We believe that this information should not be maintained by the file service, but rather provided by the components interfacing meta-workflow engines with the native workflow engines, for instance GEMLCA or the SHIWA pool. Native workflow engines are not aware of the file service and do not need to be instrumented; conversely, internal operations such as the handling of replicas, and of transfer to computing jobs is totally delegated to the native system. A complete use-case would also require converting exchanged files between A and B, using a pivot format as proposed in D4.1 [2].

Basic file transfer interface for users. A basic web interface can be implemented for the file service to enable basic API operations by users, such as upload, download (using method `transfer`), and transfer monitoring (using method `status`).

File transfer agents. File transfer agents can also be implemented to enable asynchronous transfer operations from user workstations. For instance, an agent could be implemented to upload files of a particular user directory to a remote directory on a specific DCI. Transfers could be started using method `transfer` and their identifiers stored in a local database. Depending on the transfer status, the agent could restart or cancel them. A similar principle can be implemented to download workflow results.

4 Conclusion

We presented an architecture to facilitate the handling of data transfers in meta-workflows running on different DCIs. A file service is used to adjust DCI paths in order to provide a unique indexing scheme. It can be accessed through a RESTful API providing methods to generate URIs, transfer files and monitor file transfers.

The architecture described in this document has not been implemented yet. Globus Online [3][12] can be considered to meet some of the requirements listed in this document. The SCI-BUS Data-Bridge is a good candidate to implement it as it already provides several functionalities described here. In comparison, Globus Online [3] and the EMI File Transfer Service[13] are only interfaced with a limited number of storage backends.

[12] http://www.globusonline.eu/
[13] http://www.eu-emi.eu/emi-2-matterhorn-products/-/asset_publisher/B4Rk/content/fts-1

References

[1] ER flow project consortium. Data interoperability requirements of applications. Technical report, ER-flow FP7-Infrastructures-2012-1, RI-312579, 2013.

[2] ER flow project consortium. Virtual data objects specification. Technical report, ER-flow FP7-Infrastructures-2012-1, RI-312579, 2013.

[3] Ian Foster. Globus online: Accelerating and democratizing science through cloud-based services. *Internet Computing, IEEE*, 15(3):70–73, 2011.

[4] Jianwei Ma, Wanyu Liu, and T. Glatard. A classification of file placement and replication methods on grids. *Future Generation Computer Systems*, 29(6):1395–1406, 2013.