European Data
Grant agreement number: RI-283304

D7.5.2: Technology adaptation and development framework (final)

Author(s)	Emanuel Dima (EKUT), Christian Pagé (CERFACS), Reinhard Budich (MPI-M)
Status	Final
Version	v1.0
Date	28/04/2015

Abstract:

This deliverable is reporting on the progress of the construction and integration of the Generic Execution Framework (GEF), as well as additional required tools and components. The GEF is a mechanism designed for the enactment of scientific workflows on massive amounts of data in an environment where the data is readily accessible. This document describes the design and current state of the GEF platform and how existing EUDAT user technologies have been incorporated in its design.

Document identifier: EUDAT-DEL-WP7-D7.5.2	
Deliverable lead	STFC
Related work package	WP7
Author(s)	Emanuel Dima (EKUT), Christian Pagé (CERFACS), Reinhard Budich (MPI-M)
Contributor(s)	Stephan Kindermann
Due date of deliverable	1/03/2015
Actual submission date	28/04/2015
Reviewed by	Stéphane Coutin (CINES), Luca Trani (KNMI)
Approved by	PMO
Dissemination level	PUBLIC
Website	www.eudat.eu
Call	FP7-INFRA-2011-1.2.2
Project number	283304
Instrument	CP-CSA
Start date of project	01/10/2011
Duration	36 months

TABLE OF CONTENT

LIST OF FIGURES

1. INTRODUCTION

Many scientific disciplines face the need to process vast amounts of data near-line to their storage location. This need is driven by the exponential increase in the size of available data and the subsequent cost of moving it to single or multiple remote computing facilities. Another strong need is to share and inter-compare scientific data among researchers which creates the requirement for central and federated repositories. The computer-assisted research is reaching an inflection point, where moving the data becomes a significant part of the cost and length of data processing[1].

In the course of the EUDAT project massive data transfers were found to be a typical part of many scientific workflows. This was the reason behind developing the Generic Execution Framework (GEF), designed to decrease the need for data transfers between storage and computing locations by running common data processing tasks in the data center, near data.

An important design requirement for the GEF is to be useable as an integrated component in the scientific workflows of user communities. At the same time, the GEF must be able to execute the workflows the user communities already have, close to the source data. These requirements assign considerable importance to the programmable interface (API) of such a framework. The API determines how easy it will be for the GEF to be integrated in workflows or be used as enactment platform for workflows. The API is also, in practice, shaping the execution paradigm. Although the GEF will have a default web-based user interface, it is expected that it will be mainly used programmatically through the use of its API.

This document gives an overview of the current GEF architecture with its execution model choices and describes the available options in designing the API frontend of the GEF. Section 2 presents an overview of the GEF and the vision that drives its development while section 3 describes its architecture. In section 4 we are discussing the API choices, followed by a community use case (section 5) and ending with the summary (section 6).

[1] R. Balasubramonian et al., “Near-Data Processing: Insights from a MICRO-46 Workshop,” Micro, IEEE, vol. 34, no. 4, 2014. Available at http://cseweb.ucsd.edu/~swanson/papers/IEEEMicro2014WONDP.pdf

2. OVERVIEW

The main requirement driving the development of the GEF is to enable the enactment of existing community workflows, with minimal modifications to the workflows themselves. Following this requirement, the GEF has been designed initially – during the first half of the EUDAT project – to execute a restricted set of scientific workflows. Specifically, only the workflows created in frequently-used workflow management systems (WMSs), e.g. Taverna, Kepler, VisTrails, were compatible with the initial design. These workflows were to be exported to files in various formats and enacted by the GEF in virtual machines. This decision had the downside that it imposed a set of tools (the WMSs) to the user communities for using the GEF, and didn't accommodate for the communities that were using custom tools or lesser known WMSs. At the time, however, this was considered to be a necessary trade-off.

A new technology, emerging rapidly during the last two years, has changed the previous assumptions. This new technology is the Docker[2] platform for application containers, which has evolved from a new open-source project (March 2013) to a stable version 1.0 (June 2014) and now to a mature version 1.5 (March 2015). Docker is the first and currently predominant solution in the space of Linux application containers. Essentially, an application container is a virtualization solution for a single application with negligible performance overhead. The advent of application containers has had a big impact in the commercial space, with most cloud providers already providing support for executing Docker containers.

The application containers technology made possible to remove the GEF dependency on particular workflow management systems and re-base the GEF execution model on Docker containers. The new platform provides a technical solution for encapsulation of the existing tools, as they are routinely used in user communities. The result of the encapsulation process is a container image that can host the entire execution environment of a computational tool, including various libraries (sometimes requiring specific versions), additional services and data models. The container image can be moved as a single entity and executed as such on the GEF platform. This execution model also supports the enactment of workflows defined in a workflows management system (WMS), because the entire WMS (or just of the enactment component of the WMS) can itself be embedded in a container.

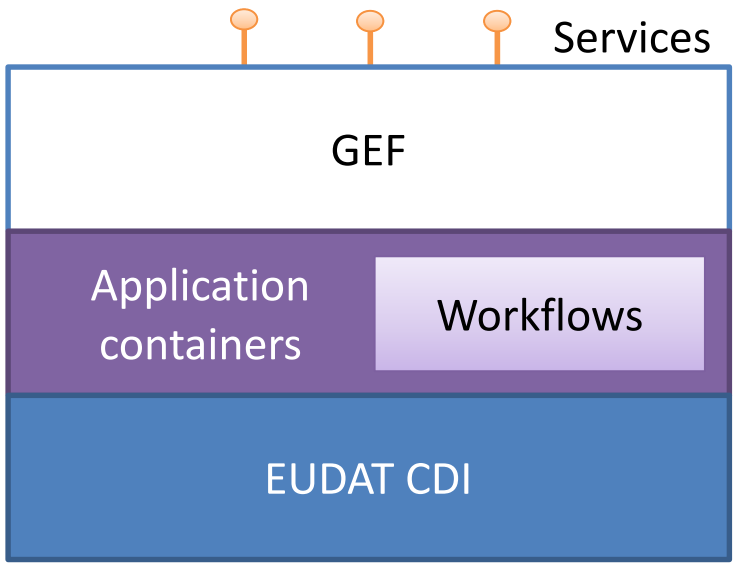


Figure 1: GEF overview

[2] https://www.docker.com/

3. ARCHITECTURE

The GEF platform contains the following software components:

1. the GEF web service
2. the executor
3. various backends

The first component is a web service that has the primary role of communicating with the users (human or software agents). The web service receives HTTP requests, parses and interprets them according to the GEF API, communicates with the executor asking for information or issuing execution requests, and returns relevant information to the user.

The second component, called *executor*, has the role of selecting an appropriate backend for a specific request, preparing the data and the required execution task, and supervising the actual execution of this task by communicating with the selected backend. The backend selection is done based on the API request type (e.g. map-reduce requests must be mapped on map-reduce backends) and service configuration files. These configuration files are specific to each deployment site and encode, for example, the difference between development and production sites.

The GEF API, in a production site, will require both a map-reduce capable environment and a workflow-enactment capable environment. Separating the job preparation stage and the actual execution supports such requirements: the previously described *executor* does the job preparation, but the actual job execution is enacted by various backends.

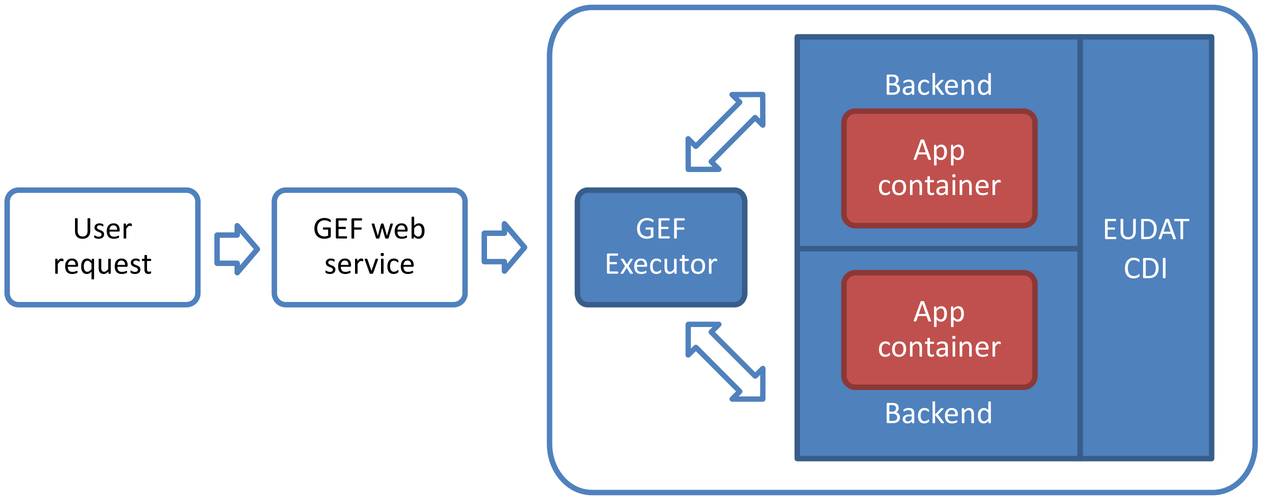


Figure 2: GEF architecture

3.1. Execution backends

There are two types of execution requests that the GEF currently accepts. The first one is the execution of a user workflow, which is encapsulated in a Docker container as described above. The second type of request is a map-reduce request, in the form of a PigLatin (see below) script.

For the first type of request the range of possible backends stretches from a simple Docker server to a cluster manager with support for Docker containers (e.g. Apache Mesos[3]). The simple Docker server is a good choice for development and is currently the only one implemented and supported. Moving to a production environment the need will arise for a more complex environment, capable of scheduling the use of resources. Despite the fact that Docker is a very new technology, there are already multiple solutions for such a backend; we think three of them are of particular interest:

Docker Swarm[4] creates a single logical Docker server distributing its load over multiple hosts. As it exposes the standard Docker API it provides a simpler upgrade path from the single host development backend. The goal of the project is to provide swappable backends, possibly making it compatible with a full-fledged cluster manager like Apache Mesos.

Kubernetes[5] is another Docker management solution. Kubernetes uses the Docker container as an abstraction unit and builds on top of it. It defines a Pod as a collection of containers that together make a logical entity and support the execution of a service or application. Kubernetes also takes care of the resource scheduling and the management of the entire life cycle of a container.

Mesos is a large-scale cluster manager with support for Docker containers. Mesos is particularly interesting for the GEF because it can also host a Hadoop environment, either directly or by installing the native Hadoop YARN resource manager on top of it (the Myriad[6] project).

The second type of request that the GEF accepts is a map-reduce task expressed in PigLatin. In this case, the choice of a backend is obviously Apache Pig[7] on top of one of the Hadoop[8] distributions. The resource scheduling requirement in this case would be resolved by the underlying resource manager, YARN by default.

3.2. Specification and implementation

The GEF development is currently done on two levels. The community requirements and technical limitations drive the overall design and the API of the platform, which are formalized into a specification document. This specification in turn should drive the reference implementation of the GEF prototype and the insights gathered during the implementation should be used in a feedback loop to adjust the specification. A specification document is important as it provides a high level view of the platform, serves as base for testing and alternative implementations, and is also the root of user documentation.

The changes of the GEF execution model, driven by the advent of application containers; and the discussions on migrating the API from a pure HTTP to a WPS format (see below) only allowed to implement a basic prototype of the GEF. The current implementation follows the architecture described above, with the (current HTTP) API requests being forwarded to an executor module and resolved in a simple Docker container backend that interfaces with the B2SAFE[9] storage.

Both the specification and the prototype implementation are publicly accessible and can be currently found on GitHub at https://github.com/GEFx.

[3] http://mesos.apache.org/
[4] https://docs.docker.com/swarm/
[5] http://kubernetes.io/
[6] https://github.com/mesos/myriad
[7] http://pig.apache.org/
[8] http://hadoop.apache.org/
[9] http://www.eudat.eu/b2safe

3.3. Data access and interaction with EUDAT services

The GEF, as part of the EUDAT environment, closely interacts with the suite of existing EUDAT services. This is accomplished through the use of PIDs. The PIDs are required at the API level to indicate the primary data sets used in computations. One of the tasks required from the GEF is to resolve the PID and provide access, in the execution environment, to the actual data set indicated by the PID, but without cloning the full dataset locally.

A PID is an indirection layer with associated metadata. The PID points to a resource at a certain URI, and additionally can contain checksum information of the data or, if the PID links to a replica, the URI of the original object. The data set URIs, that the PIDs point to, are very diverse in practice, and programmatically they cannot be treated uniformly. We encountered the following cases:

1. PIDs that point to B2SAFE data objects (currently iRODS identifiers). The underlying data sets can be in this case directly accessed with specific commands and libraries, if the user has the proper access rights.
2. PIDs pointing to human-targeted HTML pages. The solution to access the data in this case is to scrape the HTML page and retrieve the links to the actual data. This task, however, is dependent on each provider, because HTML pages coming from different providers have different structures, and is very precarious. This case currently occurs for B2SHARE-stored data sets, but the service will soon change the PIDs to point either directly to data or to a formal metadata document[10].
3. PIDs pointing to machine-readable documents. Just as in the previous case, getting access to the data is document specific and depends on the data provider. An example for this case is the CLARIN community PIDs, which point to a formal CMDI document (a metadata document in XML format).

The data objects managed by B2SAFE are the primary data sources for the GEF. B2SAFE provides the means for storing very large data sets, which the other storage services are not designed to do. But for a regular user, the GEF interactions with B2DROP[11] and B2SHARE[12] are also very important. B2DROP can become a versatile storage service for temporary data, caching data processing results – which could be further analyzed using GEF – or accumulating results for later inspection. B2SHARE can be used as the destination for data sets produced by a workflow, assuming that the workflow is well-tested and produces results deem for archival. In this latter case the tangible result of the workflow will be the PID of the resulting data set, which is automatically created by B2SHARE.

[10] https://github.com/EUDAT-B2SHARE/b2share/issues/593
[11] http://www.eudat.eu/services/b2drop
[12] http://www.eudat.eu/services/b2share

4. APIS

Designed as a web service available over the common HTTP protocol, the GEF was initially provided with a simple REST API for data manipulation and data processing. The data management functions were from the beginning meant to be temporary (used during development) and eventually replaced with functions of the EUDAT HTTP interface (e.g., the upcoming B2STAGE HTTP interface[13] and the B2SHARE REST API[14]). More recently, the wide uptake of WPS[15] in the geophysical community prompted us to reconsider the interface specification for the second API functionality, regarding data processing.

The basic data processing functions of a GEF service are:

- execution of predefined operations on datasets specified by PIDs
- the ability to extend the range of available operations by uploading user defined workflows and tools (encapsulated in an application container).

WPS was conceived as a way to make GIS calculations available on web. Nowadays it is also used as a generic standard interface to expose processing services to the web; it is part of a large set of OGC service specifications covering e.g. service catalogues and visualization. A WPS service also exposes a range of operations, which can be inspected and executed on data provided by the user. It is based on very generic and barebones specifications. These specifications ensure that the interface can support the communities' current and future processing services, but also benefit from existing tools that adhere to the standard interface.

The following sections discuss and compare the two interface specifications.

RESTful API

The GEF API is organized as a classic RESTful service, each functionality being a resource and available at a unique URL. Such a resource can be interrogated for metadata or applied on other resources (i.e. datasets). This API has the following basic functionalities:

- Data upload (serving as input for other operations) and retrieval, to be deprecated as stated above
- Execution of the predefined filter and mapreduce functions; the filter operation filters a specified data set using a (community specific) query language to specify the operation. The mapreduce applies a transformation (e.g. specified in Pig Latin16) on the specified data set using a mapreduce framework.
- Workflow management service: GEF authorized users can upload parametric workflows that become full functions of the service.

WPS

WPS is organized differently than GEF. A WPS service only exposes one URL. The requested service operation must be sent as a parameter of the HTTP method. WPS has a fixed number of operations used for discovery and execution of functions:

[13] http://www.eudat.eu/b2stage
[14] https://b2share.eudat.eu/docs/b2share-rest-api
[15] The spec: http://www.opengeospatial.org/standards/wps
A more approachable, simple introduction: http://docs.geoserver.org/stable/en/user/extensions/wps/index.html
[16] http://pig.apache.org/

- *GetCapabilities*: this operation returns an XML document containing a list of the service functions (the service capabilities)
- *DescribeProcess*: this operation requests a process identifier (which can be found in the response from the *GetCapabilities* operation) and returns metadata about the process and its inputs and outputs.
- *Execute*: this operation is a request for the service to start a process, where user specifies the process identifier and the data inputs. The input data can be directly provided in the request or identified by a URL.

The WPS interface does not prescribe a way to implement WPS service chaining and generic workflow support. Various existing workflow engines can be integrated e.g. as a WPS service allowing the upload of generic workflow descriptions as a processing request. See e.g. the Birdhouse[17] WPS component system currently supporting *dispel4py* as well as RestFlow workflows.

Very recently the new WPS 2.0 specifications have been published (2015-03-05)[18]. This is a big step in WPS specifications and functionalities, as the new features are essential to the GEF interface[19]:

- More modular (by applying the core-/extension-model)
- Support asynchronous processing
    - Abort
    - Pause
    - Resume
    - GetStatus
    - MaxWaitTime
- Support more data types, esp. OGC web services
- Easier to implement
- Better suited for Grid-computing

4.1. Discussion

There is a great deal of similarity between the two options, as both APIs are designed to describe and execute operations implemented by a service instance, using HTTP as a communication layer and moreover:

- Both APIs support asynchronous processing, where the results of an operation are stored locally and retrieved by the client in a subsequent request
- Both WPS and RESTful GEF are simple and abstract interfaces to processing functionality. Implementations of these interfaces should both expose a clear modular “internal” processing plugin interface, which all processing implementations have to adhere to. Supporting the same internal plugin mechanism processing implementations could easily be exchanged between WPS and GEF deployments.
- Both WPS and RESTful GEF are agnostic of the underlying infrastructure to support parallel data processing. The question whether a parallel service implementation shows up as a specific type of API function (like the mapreduce function supported by GEF) can be flexibly answered depending of the use cases.

[17] http://bird-house.github.io/
[18] OGC® WPS 2.0 Interface Standard http://docs.opengeospatial.org/is/14-065/14-065.html
[19] https://www.ogf.org/OGF28/materials/1971/OGC-OGF-Session1-4-kiehle.pdf

- Both APIs are allowed to refuse processing of data for various reasons. If the GEF decides that the input data is specified by a PID or URL which makes the data access difficult (maybe requesting data transfers that would defeat the purpose of the framework), the request can be refused and an appropriate message can be returned to the user by both the RESTful API or WPS as a standard error.

The major differences between the two are:

- GEF is designed as a RESTful service, exposing resources as URLs. WPS adheres to some REST principles, but not all of them[20]. First version of WPS has been standardized in 2007.
- WPS is much heavier than GEF. An alternative for WPS that mitigates this problem is WPS-Simple[21], which removes some complexity.
- WPS has the ability to orchestrate multiple operations in a single call. A WPS process can use as its input the output of a preceding process (process chaining). GEF currently has no such functionality.
- WPS is a proven standard with multiple implementations and tools, while GEF is a design concept with a partially implemented prototype.

4.2. WPS in the EUDAT GEF context

In the EUDAT context, the goal of the GEF is to orchestrate and process large data volumes near the storage to perform data reduction (figure 1). WPS, given its large user and implementation bases, is a strong candidate to act as the EUDAT interface for the remote execution of workflows. Ideas and frameworks of the first GEF draft protocol can then be implemented and integrated within a WPS backend.

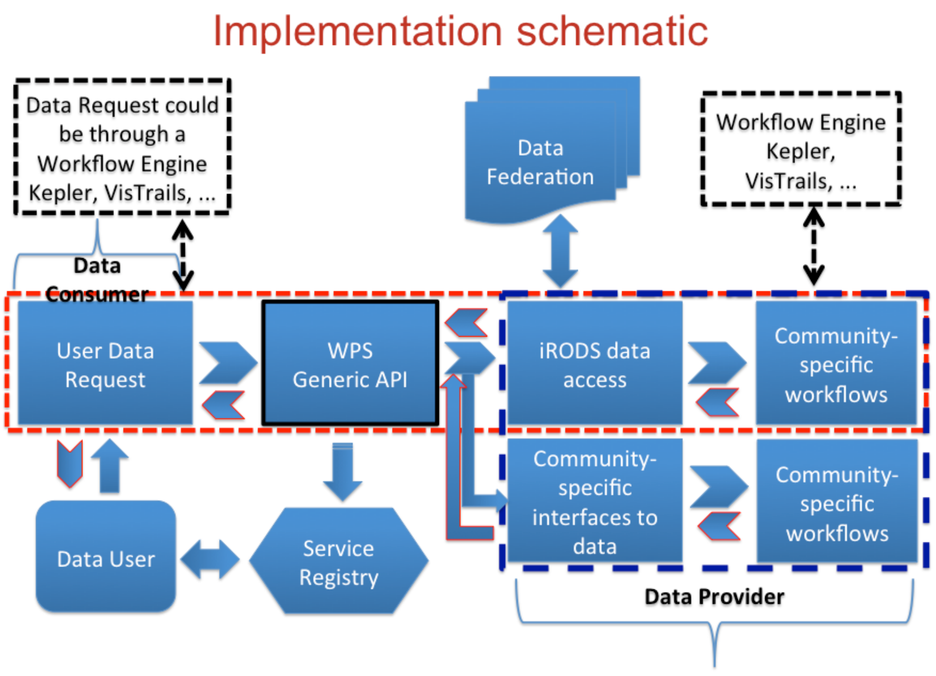

Implementation schematic
Data Request could be through a Workflow Engine Kepler, VisTrails, ...
Data Federation
Workflow Engine Kepler, VisTrails, ...
Data Consumer
User Data Request
WPS Generic API
iRODS data access
Community-specific workflows
Community-specific interfaces to data
Community-specific workflows
Data User
Service Registry
Data Provider

Figure 3: GEF Implementation Schematic

[20] See *Assessment of OGC Web Processing Services for REST principles*, at http://arxiv.org/pdf/1202.0723v2.pdf
[21] http://geoprocessing.info/wpsdoc/1x0Simple

The EUDAT GEF architecture is schematically shown in figure 2. The WPS in that framework is acting as a protocol interface to either EUDAT data access services or community-specific federations interfaces. WPS in that sense is acting as an abstraction interface to multiple federations from heterogeneous communities, which enables EUDAT services to be accessed and combined with cross-communities federated data. The difference to the first draft of GEF is that instead of developing a new protocol from scratch based on RESTful http, the WPS protocol is used instead. But the inner framework of the GEF draft design remains exactly the same with identical functionalities, as shown in figures 3 and 4.

The first implementation of GEF/WPS will need to develop and specify the following:

- Interface to EUDAT services:
  - Metadata Services (B2FIND)
  - AAI (B2ACCESS)
  - Catalog Services
  - PID system along with search services
  - Data staging (B2STAGE)
  - Data management (B2SAFE), and its integration with map-reduce tasks
- Community specific interfaces to their data federation interface, such as ESGF for ENES, WebLicht for CLARIN, etc.

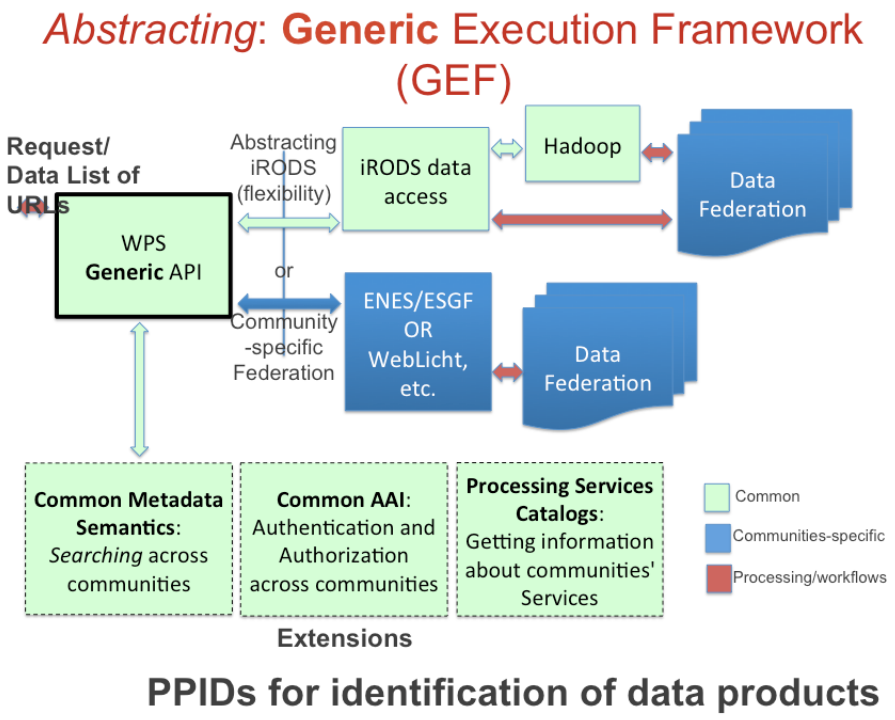

Abstracting: Generic Execution Framework (GEF)
Request/ Data List of URLs
WPS Generic API
Abstracting iRODS (flexibility)
or
Community -specific Federation
iRODS data access
Hadoop
Data Federation
ENES/ESGF OR WebLicht, etc.
Data Federation
Common Metadata Semantics: Searching across communities
Common AAI: Authentication and Authorization across communities
Processing Services Catalogs: Getting information about communities' Services
Common
Communities-specific
Processing/workflows
Extensions
PPIDs for identification of data products

Figure 4: GEF Framework with WPS backend

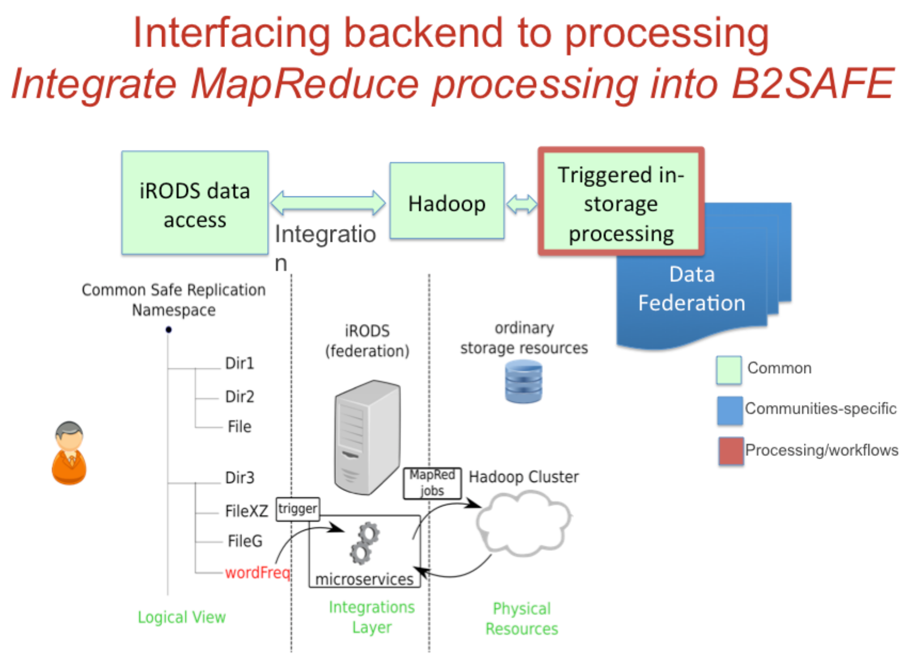
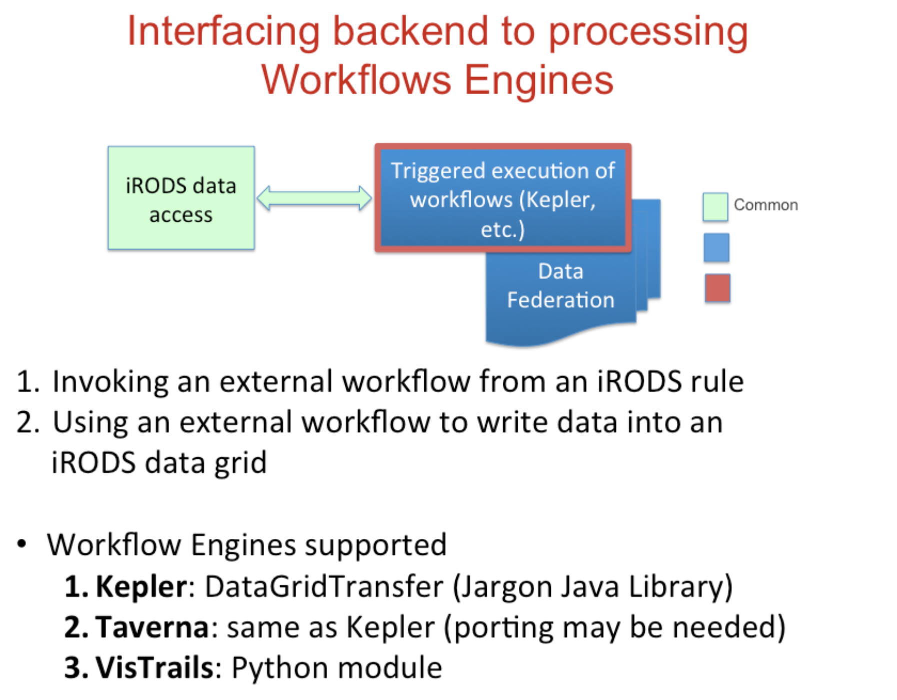

Interfacing backend to processing
Integrate MapReduce processing into B2SAFE

Figure 5: B2SAFE and Hadoop integration in the GEF framework


Interfacing backend to processing
Workflows Engines

1. Invoking an external workflow from an iRODS rule
2. Using an external workflow to write data into an iRODS data grid

- Workflow Engines supported
  1. **Kepler**: DataGridTransfer (Jargon Java Library)
  2. **Taverna**: same as Kepler (porting may be needed)
  3. **VisTrails**: Python module

Figure 6: GEF integration of Workflow Engines

The self-describing operations of WPS allow for discovery of WPS instance capabilities and also for the development of automatic GUI clients.

5. USAGE SCENARIO

To demonstrate the potential usage of the GEF we present a possible use case, taken from the linguistic domain but valid for other research domains as well.

In the linguistic research domain large textual corpora are collected, augmented with linguistic analysis, and then archived as a digital object in institutional repositories. The new repository object is then given a PID and shared for usage in other research activities. Such a digital object often serves as a primary source of data for a research group. It is often the case, however, that only a small subset of the entire data set is relevant for the research purpose. Therefore, the data set must first be filtered. When the size of the source data set is very large, the filtering process becomes expensive both in terms of time and storage space. The process can become even more expensive if the source data set is stored compressed.

The benefit of community integration with the EUDAT CDI is evident in this case. The institutional repository that hosts the research digital objects would be replicated to a EUDAT data center via B2SAFE. The filtering phase would be executed, using the GEF, in the same data center, without costly, lengthy network transfers. The use of application containers would allow transparent utilization of any filtering tool required by the task. The resulting data, likely much reduced in size compared to the original data set, would then be temporarily stored and available for download, possibly by using the B2DROP or B2SHARE services.

6. SUMMARY

The GEF meets the requirements of enacting community workflows close to the data by using the Docker application container platform. The software tools used in practice by communities can be packed into containers, which can be easily transported close to the data and executed. The GEF platform provides a consistent API for transporting and deployment of containers, as well as for running the containers against datasets identified by EUDAT PIDs.

Our vision of using the GEF in scientific workflows is to have a repository of application containers, immutable, identifiable, and citable through PIDs, used in workflows orchestrated by any external scripts or workflow management systems. The application containers, containing the processing logic, together with the primary research data should be documented, archived, and available for inspection and execution at any time. The GEF platform will thus greatly enhance the reproducibility of scientific experiments and ease the management of data provenance.

During the next phase of this project, EUDAT2020, the GEF development will continue, and it is expected that the GEF will become a full EUDAT service.

ANNEX A. GLOSSARY

AAI	Authentication and Authorization Infrastructure
API	Application Programming Interface
CLARIN	Common Language Resources and technology INitiative. An ESFRI project in the Social Sciences and Humanities domain.
CMDI	Component MetaData Infrastructure A framework to describe and reuse metadata blueprints
ENES	European Network for Earth System Modelling
EPIC	European Persistent Identifier Consortium
GEF	Generic Execution Framework
PID	Persistent IDentifier
REST	Representational State Transfer
iRODS	Integrated Rule-Oriented Data System
WMS	Workflow Management System (e.g. Kepler, Taverna, VisTrails)
WPS	Web Processing Service. An OGC interface standard for invoking processing services.