WissGrid Documentation

Work Package 3 - Long-term Preservation of Research Data

Integrating Fedora Commons and iRODS[1]

Authors	Work package 3 – Long-term preservation and research data
Editors	Panzer, Samadi
Date	13.04.2012
Document Status	Final Version
Document Version	1.0

[1] This document was created by the WissGrid project with support from the German Ministry of Education and Research (BMBF).

Table of Contents

1 Introduction

The WissGrid project[2] Work Package 3 (AP3) deals with questions of long-term archiving, from simple, stand-alone data archives to the integration of (virtual) research environments with repositories. A software stack for long-term and trustworthy administration[3] of large datasets is being developed as part of AP3 which can be used by subject communities, installed in D-Grid and customized to individual needs.

A key challenge in developing a long-term archiving architecture is that although use cases share the need for long-term administration and storage of data in a repository, they differ significantly in their use and integration of this data. Because of the wide variety of specialist user communities and requirements, the software has been characterized by means of user profiles, which correspond to different integration variants. These in turn cover the majority of the expected use scenarios. The software stack developed in AP3 supports user profiles described below and the associated integration possibilities in accordance with the workflows set out in the specifications for a generic long-term architecture[4] and grid repository.[5]

- "Repositories as archive-backend for the grid" (aka Profile A)
  The core requirement of a repository under this profile is the support of grid workflows (grid jobs). Data objects are primarily created and edited in the grid which makes protocols to bind the repository to the grid a key requirement for this profile.

- "Data grid as storage for repositories" (aka Profile B)
  In this profile the core requirement is binding the repository to a user environment. Data objects are created and edited in an interactive, collaborative user environment. The support of standard APIs and communication interfaces to permit interaction between the user environment and the repository is a key requirement for this profile.

The chapters following next will describe the features and attributes of these two profiles in more detail.

1.1 Repositories as archive-backends for the grid (Profile A)

This integration variant is applied in cases where data objects (files) saved in a repository need to be integrated directly, simply and efficiently as possible into grid jobs or grid workflows. The key issue here is the ability to reference objects with the help of persistent identifiers (PIDs) and the support of grid compatible communication mechanisms, i.e. the support of grid protocols for import and export (e.g. GridFTP, iRODS iCommands). The grid repository specification defines this user scenario as Profile A.

---

[2] http://www.wissgrid.de

[3] The term administration encompasses duties which go beyond those assigned to CRUD operations such as the creation and maintenance of metadata, relationships or indexes, the implementation of aspects of bit preservation and the allocation of the requisite interfaces.

[4] Generic Long-term Archiving for D-Grid. WissGrid, 2010

[5] WissGrid Specification: Grid-Repository. WissGrid, 2010

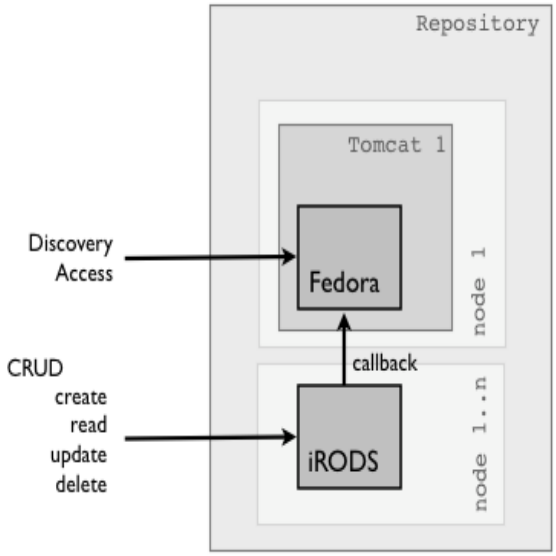


Fig. 1: Repositories as archive backend for the grid

Community requirements: The *creation and editing of large datasets in the grid* with persistent availability and trustworthy retention over long periods of time.

User scenario: Grid jobs draw their data and write their output directly into the repository. An explicit "staging" of data in advance of editing and transfer to a service is not applicable since these are being referenced via PIDs or via iRODS iCommands and can be integrated into the job description (or job execution).

Technical implementation: In the grid repository specification iRODS[6] is defined as a system for its technical implementation. iRODS is a rule based data hosting and storage system. Because of its limited interface options it will often be useful to place a Fedora[7] server in front of an iRODS installation so that it can benefit from its functionality. Fedora is a data and metadata management system which offers features for administering data objects, metadata and relationships; it permits the organization of objects and their classification; it permits the creation of virtual representations (i.e. derivative objects) at runtime and offers a larger set of interfaces than iRODS. In a cooperative System, consisting of iRODS and Fedora, Fedora serves in effect as a metadata catalog and export interface for iRODS and supports functions for content preservation and data curation tasks. iRODS in turn handles certain aspects of bit preservation, e.g. carrying out integrity checks.

1.2 Data grid as storage for repositories (Profile B)

This variant is applied in cases where the integration of repositories of with applications takes precedence, for example in conjunction with (virtual) research environments. The grid repository specification refers to this as Profile B and is derived from use cases where the grid (specifically, grid storage resources) are employed to store or preserve objects in a trustworthy manner. In the context of this integration variant objects are created in predominantly interactive environments and edited collaboratively. To enable this interfaces are required in the repository based on open, standards based technologies (e.g.

[6] Integrated Rule-Oriented Data System
[7] Flexible Extensible Data Object Architecture

REST, SOAP or JMS) which will permit simple and comfortable connections to existing user environments. Interoperability with other applications is a key requirement.

Community requirements: Here too the administration of large datasets is a significant requirement in particular of (many) small objects and their related (or derived) access forms (e.g. XML data or image annotations) which need to be administered. Over and beyond this it's also necessary from a user perspective to define custom object models (in Fedora terminology: content models), assign object metadata and define relationships between objects.

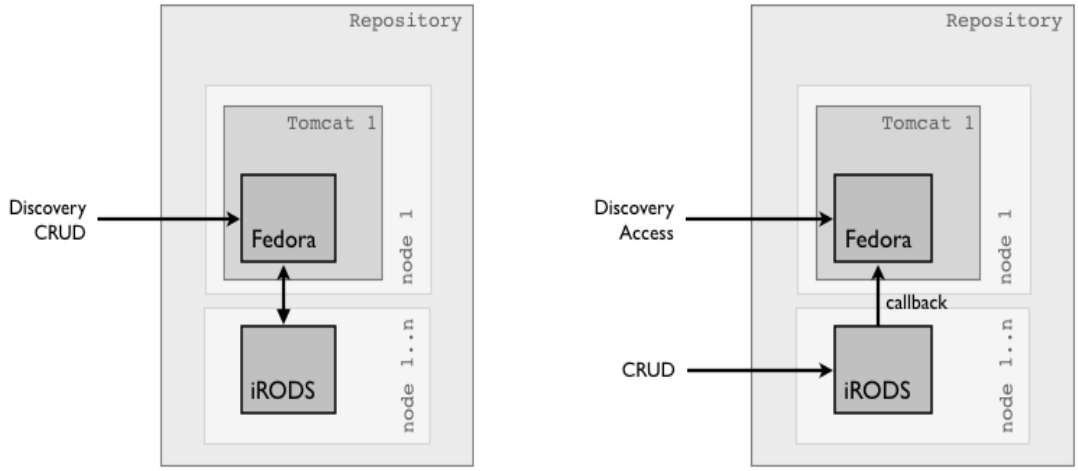
Use scenario: Data is created or edited (collaboratively) in an (interactive) user environment.

Technical implementation: The grid repository specification already lists a combination of iRODS and Fedora as a technical implementation.

In this integration variant the two options depicted in Figure 2 can be distinguished by the visibility of the iRODS server and differences in the form of access:

> a) Fedora with a *transparent* iRODS as backend, that is, access takes place exclusively over the Fedora web-interface. The use of iRODS as a storage solution remains hidden from the user.
>
> b) Fedora with a *visible* iRODS as backend. Here direct interaction with the iRODS system is possible, that is, access can also take place directly on the iRODS system (via iCommands[8] or the Jargon-API[9]). This can be desirable in order to, for example, transfer entire directories with one command (mass-ingest[10]) or to make use of the optimization features (parallelizing) offered by iRODS during transfers.

[8] Command-line tools for interacting with the iRODS system
[9] Java API for interacting with the iRODS system
[10] In an archival context "Ingest" is defined as importing data into the repository.



a) Access to Managed Content b) Access to Referenced Content

Figure 2: Data grid as storage for repositories

For our purposes this distinction will be set aside since the differences are not in the technical implementation but merely in the manner in which the system is used. The configuration remains the same in both cases.

1.3 Overview of system components and integration aspects

For the most part, the system configuration consists of: Fedora, iRODS, the `wissgridservice` module, various Fedora plugins and Fedora based services. To meet user requirements Fedora and iRODS are used in combination in the previously described combinations in order to make the most of their respective strengths in a cooperative environment. The two systems, Fedora and iRODS are developed as autonomous systems in a user environment that assumes that applications will be built on top of them. On account of their integration in AP3 for the repository the two systems need to exchange data, i.e. constantly synchronize their current datasets. The mechanism by which the synchronization between Fedora and iRODS is established are called Callback[11] and Forward[12] (see callout on p. 11).

Several components as well as customization were developed to integrate Fedora and iRODS. The most significant of these in the context of AP3 in Fedora are the iRODS-Akubra plugin which allows Fedora to use iRODS are a storage resource, and in iRODS the extension and customization of rules as well as the development of microservices to capture and synchronize events (e.g. ingest or delete). Besides this a SOAP based service called `wissgridservice` was developed which is responsible for additional indexing tasks, versioning, metadata provisioning and related tasks.

The iRODS Akubra module makes it to possible for the Fedora storage layer to store managed content in iRODS. Without reaching too far ahead into the material which will be

[11] Calls on significant events in iRODS in order to ensure their integrity in Fedora
[12] Like a Callback but originating from Fedora

covered in the coming chapters, it suffices to say here that *managed content* refers to data objects which are under the control of Fedora, that is, stored and physically located on the Fedora system. In contrast to this *referenced content* is not managed by Fedora. It is merely referenced by Fedora via a URL and is stored externally. For more information on content types and control groups see Section 2.1 on p. 10.

1.4 Structure of this document

Following this general overview of different user scenarios and components, Chapter 2 will cover the implemented systems and system components, their tasks, and their interaction in more detail. Special object types which are required to represent iRODS directories and files as Fedora objects will also be introduced. Chapter 3 will discuss system configuration, the interaction of system components and possible integration variants. Questions about possible technical implementations are covered in Chapter 4 while Chapter 4 offers an overview of installation (the installation is described in full detail in the `README.pdf` in the Appendix). Chapter 6 lists what has been implemented in the current version, offers suggestions for potential improvements to the system as well as the current changelog. Chapter 7 provides entry points for developers into the system to help them better understand its operations.

Note:

The results described in this document refer to a development prototype whose suitability for production environments cannot be guaranteed.

2 System components and elements

This chapter will introduce the individual system components and their features in detail together with object types which are required to represent iRODS data objects in Fedora objects or datastreams.

2.1 Fedora

The Fedora Commons repository software is a system for data and metadata management. To this end, Fedora defines an object, a Fedora Digital Object (FDO) which acts as a container for various forms of information and represents data objects, metadata and relationships.[13] In addition FDOs offer the possibility of structuring and classifying objects abstractly (via the content model). Setting FDO specific behavior is also possible using the content model (see p. 12).

The FDO is the central component in the Fedora system. A FDO is an aggregate object, uniquely identified by a Fedora-wide PID, which stores system specific but also object-specific metadata and ultimately either incorporates the actual data objects (datastream in Fedora terminology) directly as inline XML or references these via Fedora internal IDs or external references.

The FDO is object oriented insofar as it administers data objects (i.e. files or HTTP referenced resources) and situates these into a larger context by gathering these into objects (FDOs). In turn the FDOs can be placed into a still wider context by adding relationships by means of which arbitrary, user-defined object structures can be represented. In this respect the object definition in Fedora is distinct from the iRODS context where data objects[14] i.e. files, are addressed directly. A FDO is a higher-level unit and its description is constructed with XML, the so-called FOXML[15] which in turn is defined by a XML Schema (foxml1-1.xsd).

The object properties which belong to each FDO describe administrative metadata which are used to define e.g OwnerID, State, CreatedDate, LastDate and LastModifiedDate. All other elements (shown in Figure 3) are datastreams. In principle a FDO can reference an unlimited number of datastreams although an excessive number of streams would be indicative of a poor object design. This should be avoided by properly structuring objects and by setting relationships (content-modeling).

The specific structure of a FDO is strongly dependent on its application. So, for example, it could consist:

- in the simplest case of a single datastream (data-object), or
- all digitized objects[16] in a book with a OCR file[17] and, where applicable, a TEI file with further information, or

[13] Flexible Extensible Data Object Architecture
[14] The counterpart of the iRODS data object in Fedora is the datastream
[15] Fedora Object XML
[16] Scanned photos
[17] The result of an automatic text recognition within images

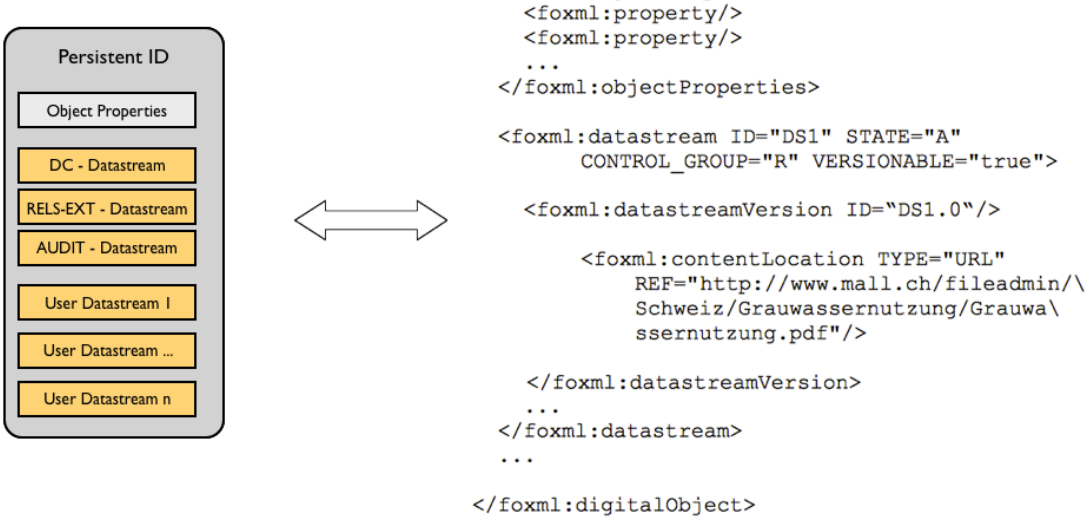


Figure 3. Relationship between FDO and FOXML

- a FDO could represent a research project for which a separate FDO would be created for each trial run or project part (in needed, separated into further FDOs) which would contain its results

In the final analysis its structure is application driven and must be defined by the relevant user communities; responsibility for this belongs to data curation.

Depending on their type and where the data objects are saved Fedora distinguishes amongst four types of datastreams or control-groups in Fedora terminology:

Internal XML Content - (X): Data (data objects) are stored directly in FOXML as inline XML (inside an appropriate tag)

Managed Content - (M): In the case of managed content Fedora is responsible not only for the administration but also handles storing the data objects via its low-level storage abstraction layer, i.e. Fedora also receives the object physically and has full control over the data object.

Externally Referenced Content - (E): In contrast to managed content all that is stored in this variant is an object referencing an externally saved data object. The physical storage is located externally and thus not under the control of Fedora. When the data object is accessed its content is "streamed" by Fedora. This option is especially useful when the user has no access rights to the storage device or to obscure the storage location.

Redirect Referenced Content - (R): Here too, as with (E) only the object reference in the form of a URL is saved. However in this variant access is not achieved via streaming

by Fedora but via URL redirect. This is particularly when deploying a streaming (i.e. media) server.

Alongside an arbitrary number of user defined streams a FDO can also make use of four part system-specific datastreams which manage administrative and descriptive metadata.

DC (Dublin Core): Each FDO includes this datastream by default, if necessary, it is generated automatically. This datastream sets the descriptive metadata which is then used by Fedora for standard searches.

RELS-EXT (Relationships-External): In this datastream object relationships are represented as inline XML. The datastream includes a RDF element which includes all corresponding relationships to this object.

RELS-INT (Relationships-Internal): In this case relationships from datastreams to objects or other datastreams are represented. This datastream belongs to the control group internal XML content, i.e. inline XML.

Audit: The audit datastream is also specified as inline XML. It records all object modifying events and on the basis of this creates an object history.

In a cooperative registry users interact with the system either via iRODS or Fedora. This interaction can occur sequentially, i.e. alternating between iRODS and Fedora. Specific actions such as an ingest can only occur via one system.

Callback: If the interaction takes place over iRODS, that is, if actions such as ingest, update or delete take place via iRODS, then these actions subsequently also have to take place in Fedora in order to ensure the integrity of the system. This is achieved via a so-called callback which is responsible for the passing the acknowledgement from the storage layer back to Fedora to keep the system synchronized.

Forward: Events in the Fedora system are passed to iRODS via a forward in order to ensure synchronization with the iRODS system.

Fedora aggregates user data via FDOs and datastreams. These aggregations have a particularly important consequence when deleting FDOs. Datastreams from the control groups internal XML content and managed content are deleted along with the FDO because they are a part of the FDO or can only be addressed via the FDO. This is not the case with referenced datastreams since these can merely be referenced. It's not possible to establish using Fedora if the referenced object has been changed or deleted. From the opposite perspective, deleting a datastream in Fedora has no effect on the referenced object. It merely gets removed from the Fedora system and remains accessible at its original location. The reason for this is that the external data object is not under the control of Fedora or iRODS.
Referenced content (in both control groups) constitutes an exception to this as it can be saved in a cooperative repository and registered back into Fedora via a callback. In this case deletion takes place via a callback initiated from iRODS, that is, the deletion is carried out in

Fedora via callback; when initiated by Fedora the deletion of a FDO or a datastream is subsequently carried out in the iRODS system via a forward.

FDOs can also be linked to other FDOs and external resources. The content model offers Fedora generic method to structure internally (type and number of related datastreams) and externally (extent of its object relations). For this purpose FDO can be related to multiple content models by means of which the structure of a FDO is fixed. In addition, a content model can define behavior, by means of which it can access FDOs. This model specific behavior is defined by its relation to service definitions (i.e. API specifications) and service deployments (implementations). In this way the defined services (operations) can be accessed from the data object and the above mentioned virtual objects or virtual representations can be created at run-time. Specifically, these are transformations of the original data objects which are being created via the defined services. Typical transformations could be, for example, a change in the format of an image file; the transformation of a XML file via a XSL stylesheet (i.e. via XSLT and XSL-FO); calculations on the basis of data objects or queries such as "deliver the nth page of a PDF document". Because of its potential to deliver virtual objects, Fedora is well suited to the task of content preservation. The role of modeling is part of data curation and needs to be carried out by the user community as these models are dependent on the concrete data objects and collections but also on community needs.

Summary: The strengths of Fedora are its format neutrality, the possibility to place objects in relation to each other (via RDF) or with metadata as well as its ability to add additional structural data as needed. Fedora permits data object versioning, the creation of virtual representations, the export of objects in e.g. the METS format, as well as connection to external systems via REST, SOAP or other messaging interfaces.

2.2 iRODS

iRODS is a storage infrastructure or data hosting system which can be used for a virtual storage system to access both logical objects and structures. Objects and structures in the context of iRODS are files and directories (i.e. data objects and collections in iRODS terminology). iRODS does not have a special object definition as is the case in Fedora. There is only the possibility to group files and group files on the filesystem and build a structure on this basis. Information about the administered iRODS objects are stored in the iRODS component iCat (catalog), a relational database. Multiple iRODS server form a zone which together share a single iCAT database. Storage and replication can (and should) bind many different storage resources together since this is also advantageous for reasons of availability and load-balance.

The system can be customized via rules (text based) and microservices (C routines) but using rules only at run-time. Rules controlled by the iRODS rule engine can react to system events or triggered at set intervals to initiate specific activities or reactions. Typical examples are the replication of different storage media (or storage resources) or calculating checksums whose regular validation (integrity tests) determines whether a damaged file should be replaced by a correct copy. In combination, rules and microservices one can be used to represent workflows.

By setting AVUs[18] on iRODS elements, that is, on data objects (files), collections (directories), resources (storage resources), resource groups, as well as iRODS users, it becomes possible to furnish these elements with metadata, and to query or edit this metadata. It is not possible to set relationships between iRODS elements directly. However unlimited AVUs can be added to iRODS elements. Relationships can then be set, for example, by having the data object store the PID (or object path) of the *parent data-object* as an AVU. The representation of relationships via RFD and the administration of relationships in a triple-store[19], with the potential to search for this, is by comparison much easier in Fedora.

In principle, iRODS can offer support for content preservation tasks by way of rule based transformations, for example during ingest, access or retrieval. If the transformation needs to be carried out continuously then this can be achieved relatively easily. But if the transformation needs to be carried out under specific conditions then putting together the requisite rules can quickly become complex. Here too, a solution based on Fedora which is able to draw on virtual representations, appears simpler.

Conclusion: The unique strengths of the iRODS system are its event driven architecture, customizable rules and microservices, implementing and controlling bit preservation (replication, integrity tests, substitution of defective copies) as well the possibility of representing workflows.

[18] Attribute Value Unit, i.e. key/value pairs
[19] A triple-store is a type of database optimized for the storage of triples. In the RDF syntax triples are formed out of subject-object-predicate groups.

	Weaknesses	Strengths
iRODS	• limited, proprietary interface • difficulty administering metadata • no relationships • no object definition • support for content preservation difficult (transformations via rules / microservices) • weak support for data curation (via AVUs)	• rule engine • support for bit preservation via rules and microservices • parallelization
Fedora	• no direct support for bit preservation (no automatic replication or integrity controls) • sequential ingest	• open and standardized interfaces • metadata administration • relationships described via RDF, saved in triple store • object definition • describe structure via content-modeling • good support for content preservation and data curation
Cooperative Repository	• User and rights administration • synchronizing stored data	• open and standardized interfaces • metadata administration • relationships described via RDF, saved in triple store • object definition • support for bit preservation via rules and microservices • parallelization

Figure 4: Comparing iRODS and Fedora

2.3 `wissgridservice`

`wissgridservice` is a standalone, SOAP based web-service which is responsible for Callbacks (see Fig. 5), object versioning and indexing in Solr. iRODs events are passed on to the service which processes these and sets `AVUs` (e.g. `PID` of the FDOs or the `DsId`[20] of data objects) for the associated iRODS elements in order to create a representation (mapping) between Fedora and iRODS objects.

[20] Each datastream is identified over a (in the context of the FDO unique) datastream ID

The `wissgridservice` acts as an arbitrator between Fedora and iRODS with the goal of reducing the links between the two systems. This allows the system to be easily used with external storage (e.g. dCache as an alternative to iRODS). In this case only the callback service which is a part of the `wissgridservice` component needs to be customized.

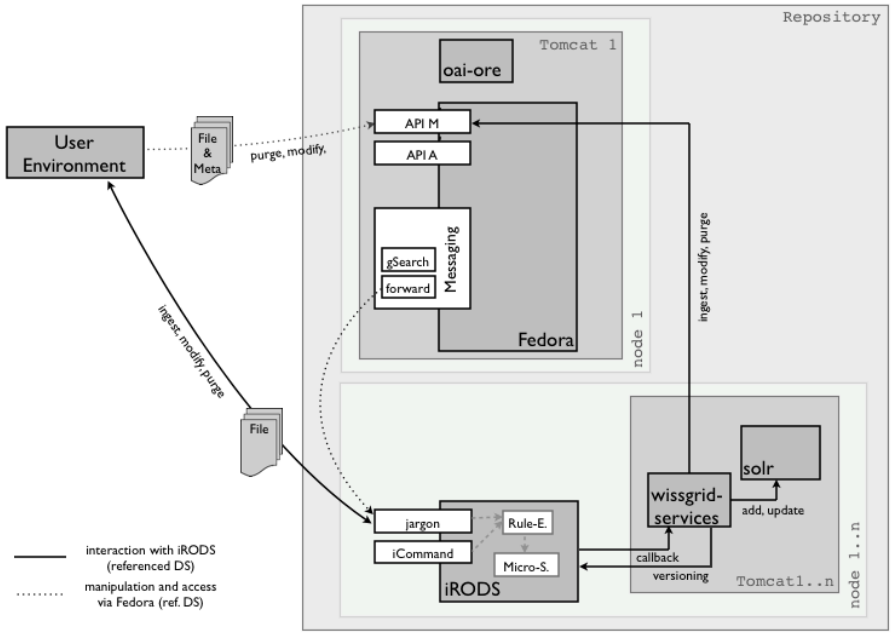


Fig. 5: Callback and Forward

2.4 Davis WebDAV Interface

Referenced content data objects are administered in Fedora via a URL reference. Since iRODS resources are addressed over the schema (`irods://...`) but Fedora only supports file and HTTP based referencing a mechanism is needed to bridge this discrepancy. The Davis WebDAV module offers the iRODS system a HTTP interface and makes it possible to access data objects via HTTP.

The data objects stored in iRODS and referenced in Fedora receive a HTTP based URL datastream location. The WebDAV component runs as a standalone service and can be installed independently of the iRODS server. It does not need to be run the same server as iRODS.

2.5 SRW/SRU, OAI-ORE, OAI-PMH and Resource Index

The interfaces offered by the Fedora server can be extended by plugins or external services. In the WissGrid context the OAI- and ORE-provider, GSearch, Pazpar2, Resource Index and the Resource Index Search Service are of particular interest.

- GSearch [20] is an external component which is tied in via the Fedora Messaging System to index Fedora objects and text based contents in datastreams. Fedora itself

only offers a basic search on the basis of Dublin Core datastreams. GSearch enables a full-text search on the content of data objects. In addition GSearch offers an interface to search queries and ties in search engines such as Lucene, Solr or Zebra.

- Pazpar2 is used for distributed search across multiple Solr indexes. It is tied in via the `mod_proxy` module of the Apache HTTP server. The indexes are created both on Fedora (via GSearch/Solr) and iRODS (via Solr).

- The basic Fedora OAI-PMH provider service offers an implementation of OAI-PMH [21]. Standard access to the metadata administered by the system is provided via Fedora's PMH interface.

- The Fedora ORE provider service offers an implementation of the OAI-ORE [22] protocol and is employed as a standalone service (web-application).

- Resource Index and Resource Index Search Service. The Resource Index is a Fedora server plugin for indexing RDF relationships. Queries to the index take place via the standalone Resource Index Search Service (RISearch) which is implemented as a web service. It currently supports iTQL and SPO as query languages. SPARQL is currently not supported.

2.5.1 Apache HTTP Server (mod_proxy)

The Apache proxy module (`mod_proxy`) is used conjunction with Pazpar2 to hide URL re-mapping and URL re-writing from users. The proxy module can provide a consistent URL format where system specific portions of URLs such as `http://<host>:<port>/fedora/` are masked or hidden. This in turn makes it possible to address the Fedora server via `http://<host>/repository/` and let `mod_proxy` forward requests on to a local or external server.

2.6 Object types and object mapping

As we have seen previously, actions can be initiated either on Fedora or iRODS, depending on how the integration has been arranged. When access is initiated on iRODS the question arises *what* should be represented on a FDO or datastream in Fedora and *how* this representation can be achieved. Three object types were defined for this purpose and implemented via the `wissgridservices` module. This allows iRODS directories or iRODS data objects to be represented in Fedora.

Single-object: Each data object is mapped to a FDO which is allocated to exactly one user defined datastream (i.e. data object). iRODS directories don't have a counterpart in Fedora.

Recursive object: Each iRODS directory is mapped to a FDO. Files inside a directory are mapped to datastreams in the FDO. Lower level directories are allocated their own FDOs which are linked recursively to higher level directories via a RDF relation `fedora:isMemberOfCollection`.

Multi-object: The extent of a FDO is variable - it can consist of a group of n data objects and a metadata file (METS, BagIt or OAI-ORE). The FDO is created during ingest when the

metadata is imported and a callback is triggered to Fedora. All other iRODS are created in the context of edits to the metadata file. The location of the associated data objects are derived from this file and added to the FDO as datastreams. It is important that all necessary files already be present in iRODS when the metadata is edited. A naming convention determines whether one is dealing with a descriptive file. The mapping configuration is determined by a property file (see 5.5 on p. 30) in which the link between a directory tree and an object type is fixed (i.e. all directories or data objects beneath a certain directory should be of the previously set object type and treated appropriately).

During ingest the allocation of datastreams to FDOs takes place implicitly when the API method `addDatastream` is given the `PID` of the associated FDO. Saving files takes place at the Fedora Storage level with very little influence by user on how these are saved (the data directory can be selected) or on how the files are structured. Fedora controls the content of the datastreams. Datastreams belong to the control group managed content which means that data objects are completely administered by Fedora. Access and changes to the data objects are only possible via Fedora (see p. 12 for information on deleting FDOs and datastreams).

3. Design, Configuration and Interaction

This chapter will discuss system configurations, the interaction of system components with each other as well as possible integration variants.

The Fedora system offers interactive access via its REST or SOAP interfaces (API-A for access, API-M for editing), in iRODS this is offered via its command-line tools (iCommands) or with the Java-based Jargon-core API. The Davis WebDAV presents a web-interface which can be accessed via a web browser, WebDAV clients or other HTTP-based mechanisms. Fedora uses the Jargon-core API to communicate with the iRODS system. The Callback, that is, the reporting back of events on the iRODS system to the Fedora system, runs over a SOAP-based web service (`wissgridservice`).

The next section will fist cover the variant *Data Grid as Storage for Repositories* (Profile B), as in this variant the repository is used from both sides (iRODS and Fedora) and hence exposes all possible integration possibilities. This in turn will significantly reduce the descriptions required to detail the second integration variant, *Repositories as Archive Backend for the Grid* (Profile A).

3.1 Integration variants: Data grid as storage for repositories

Profile B distinguishes between two sub-variants: *Fedora with transparent iRODS as backend* and *Fedora with visible (i.e. directly usable) iRODS as backend* (compare Fig. 2 on pg. 7). The differences between the two consist mainly in the way the system is used and in the awareness of users of where the parts of the system are located.

A point on which the two system differ (and under some circumstances can differ substantially) is rights and user management. Under the current scheme, rights and user management is implemented at the application level and access to both systems is carried out via robot users.[21] In the transparent iRODS variant it is possible to use the Fedora rights and user management to secure the system. Fedora would then interact with the iRODS server by means of a robot user (with admin rights).

Besides a rights and user management system, the configuration of the system will draw on the following components:

- A Fedora Server[22]
- At least one, typically however, several[23] iRODS servers, which together form an iRODS zone and which share an iCAT (metadata catalog). In addition, several storage units should be included as well, to guarantee data safety.
- The `wissgridservice` module, which can be deployed centrally or in a distributed manner. The installation should take place on the iRODS server to avoid additional transfer loads when indexing streaming data objects. The service is called on each interaction with iRODS.

[21] A special user with far reaching (if necessary, admin) rights
[22] In the case of a federation or a server cluster this can also comprise several servers
[23] On account of failure redundancy and load balancing

- The Davis WebDAV Module, which is used centrally on the Fedora server.

- Additional, optional modules, e.g. the Fedora Generic Search Service (GSearch), the OAI-PMH[24] or OAI-ORE[25] provider, the Fedora Resource Index or the Fedora Resource Index Search Service.

- The Cheshire 3 Information Framework can be installed on iRODS to equip it with interfaces to SRU[26] or OAI-PMH. However in evaluating this system its dependencies on specific versions of Ruby and iRODS were found to be cumbersome. As a result its use is currently not recommended.

In the case of this integration variant, preparing and making data objects available for access take place primarily via Fedora. Ingest and Updates of data objects however are generally handled by the iRODS server.

3.1.1 Interaction via Fedora (Managed and Referenced Content)

Data objects physically passed to the Fedora system via its REST or SOAP as files for storage on the Fedora Storage Level (see dotted workflow in Fig. 6) belong to Managed Content control group.

There is also the possibility to reference *external* data objects from Fedora (referenced content). Here one must keep in mind that the referenced datastreams are stored outside the control of Fedora (and iRODS) and that Fedora will not know if these files are deleted (see p. 12 for more information on deleting datastreams). Similarly, deleting referenced content on Fedora has no effect on the referenced data object which will continue to exist at its old address (URL).

If this behavior does not meet expectations then the addition of a datastream needs to take place either via Fedora's interfaces (together with the physical transfer of the data objects) or directly via iRODS by means of a callback. This is the only way to ensure that the data object remains under the control of the repository. A deletion in iRODS is reported back to Fedora (callback) and a deletion in Fedora is passed on to iRODS (forward).

[24] Open Archives Initiative - Protocol for metadata harvesting
[25] Open Archives Initiative - Object reuse and exchange
[26] Search and retrieval via URL

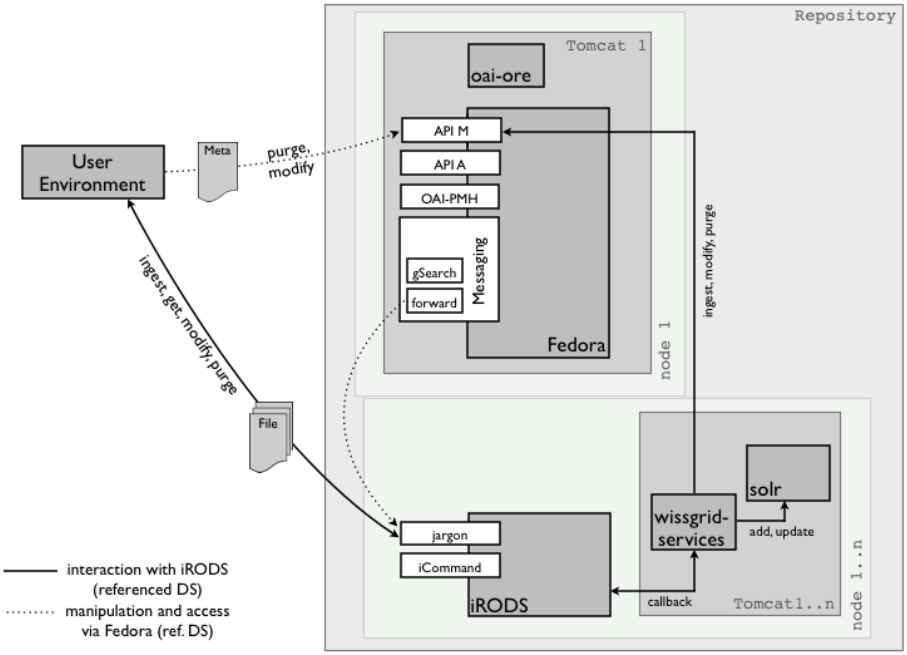


Fig. 6: Both integration variants for Profile B

3.1.2 Interaction via iRODS (Referenced Content)

When objects are added or manipulated in iRODS, a callback notifies and updates Fedora of the change (see solid line workflow in Fig. 7). The callback passes the event to the Fedora server so that it can initiate the necessary response (e.g. create a FDO, add or remove a datastream etc.). The callback is carried out via an intermediary component, the `wissgridservice`. This component is also responsible for versioning, indexing, setting metadata on iRODS elements as well as initiating the actual service call on the Fedora server.

New data objects are indexed via Solr and any changes and deletions are reflected in its index. When a new version is created a copy of the data object is placed in the versioning directory. When a current data object is deleted the next oldest copy is determined becomes the new current version.

In a typical workflow, an iCommand is sent which triggers a response on the iRODS server. As the command is being processed one or more rules are activated. A rule can activate one or more additional rules or call on multiple microservices. A microservice takes on the task of fulfilling the actual request and delegates parts of this task to the corresponding service calls in the `wissgridservice` component.

Example: `iput a.txt textdir/b.txt` is an iRODS iCommand to ingest the file `a.txt` into the current, local working directory. The ingest triggers the rule `acPostProcForPut`. This (post-) rule initially calls the microservice `msiSysChksumDataObj` which carries out a checksum calculation and delegates the further processing to the microservice `msiWissGridCallbackForPutObject`, which determines the "object path" (i.e. the

logical[27] storage location) for the data object. This microservice then calls the method `addDS` (add datastream) in the `wissgridservice` module. Here the iRODS object type is initially determined (by comparing the directory `testdir/` with the contents of the property file). Based on this the `AVU` metadata are linked to the iRODS object, versioning and indexing is carried out, and in conclusion, a service call (e.g. `addDS`) is made to the Fedora system.

Building on the example described above the next section will offer an overview of how the `wissgridservice` module handles different object types (see Chp. 2.6, p. 17):

- For single objects a FDO is created to which the data object b.txt is added as an single datastream.

  Versioning: If versioning has been activated in the property file a copy of the datastream `b.txt` with extension `._v0` is copied into the version directory.[28] If, later, the data object is overwritten a version (copy) with extension `._v1` is placed into the version directory etc. Should the current version be deleted, the data object `testdir/b.txt`, which corresponds to the current version, is replaced by the next oldest version (`testdir/_version_/b.txt._v0 -> testdir/b.txt`). If by doing so the final version is deleted (i.e. no further versions exist) the FDO is deleted.

- For recursive objects the data object `b.txt` is added to the FDO associated with the directory `testdir/` as a test stream. Versioning takes place as described above. When a new (nested) directory is created a new FDO is created automatically and linked on Fedora via RDF to the parent (directory) FDO (`fedora:isMemberOfCollection`).

- For multi-objects a FDO is created with the metadata file `b.txt` and all data objects referenced by it added to the FDO as datastreams. Versioning takes place as described above. At this time BagIt, METS and OAI-ORE are supported as metadata formats. The system assumes that all data objects referenced by an object in the metadata file are accessible in the iRODS system.

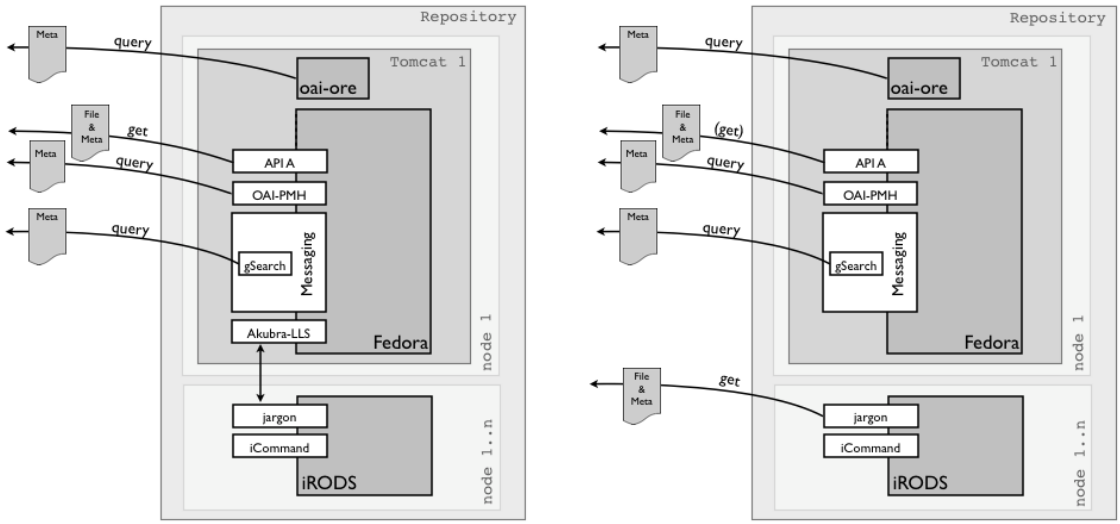
It is worth mentioning that overwriting the description file has no effect, is ignored and does not lead to a new FDO being created nor to an update of an existing FDO. However if the old descriptive file is deleted and a new one takes it place, then this leads to a new FDO since no connection exists between the old descriptive file (or can be ascertained since its deletion also removes the associated AVUs which list these connections). If versioning were permitted on descriptive data it would not be clear how to handle existing datastreams. Is the new version an update or a complete replacement of the existing FDO? Do existing datastreams belong to the new version even if they are not listed in the descriptive file or should they be deleted from the FDO? Deleting or overwriting a descriptive file is

[27] By distinguishing between logical and physical storage locations iRODS is able to limit, for example, the max. number of allowed files per directory or allocate certain data objects to defined data resources in a way that is transparent to users

[28] Each directory with data objects contains a version directory, e.g. testdir/_version_/. The name _version_ can be changed in the property file

particularly complex since all old associations (AVUs) need to be deleted and reset. For this reason this option is currently not supported.

Manipulating and deleting a FDO for multi-objects is thus only possible via Fedora (i.e. via its messaging service). Manipulating data objects can also be carried out via iRODS (i.e. overwriting and adding a new version as well as deleting a version). Adding a new datastream is only possible with Fedora (managed content) since there no iRODS element exists which could server as counter piece to a FDO. Consequently it is not possible in iRODS to directly or automatically associate a datastream with a FDO.



a) Access to Managed Content b) Access to Referenced Content

Fig. 7: Access Options

3.2 Integration variant: Repositories as Archive Backend for the Grid

As previously discussed in the Introduction, in this configuration the system can either consist of a single iRODS server or as combination of Fedora and iRODS. The first case will not be discussed further since no changes to the iRODS system are necessary (iRODS standard configuration). If the versioning system developer as part of AP3 is to be used in this configuration the `wissgridservice` component will need to be adapted. Currently this component mirrors iRODS directories and data objects onto the corresponding Fedora objects (see object types 2.6 on p. 16) and implements versioning. Since in a scenario in which iRODS is used without Fedora there are no PIDs or DsIds, a different representation of related objects must be found. In the combined configuration versions are represented in such a way that all versions belonging to a datastream share the same PID and DsId and a version capable extension of the filename or else a AVU with associated timestamp can be distinguished from each other.

For this reason we will only describe a configuration in which a Fedora server is placed in front of an iRODS server to administer metadata. In principle, this configuration is the same as Profile B (*Data grid as storage for repositories*), although, depending on one's requirements, it can be helpful to limit the interaction with the Fedora server by means of XACML policies. Restricting ingests and other changes in Fedora ensures that sufficient (iRODS managed) referenced content is available to the system.

In this application profile, interactions with data objects take place primarily via iRODS while Fedora can be viewed as a metadata catalog for iRODS and as a means of extending its limited external interfaces. Even in combination with the Fedora server the core task of this profile is to support Grid workflows, i.e. efficiently embedding data into grid jobs.

3.2.1 Interaction via iRODS

A significant characteristic of this integration variant is that it only administers referenced content since nothing else can be directly tied into grid workflows. Referencing managed content via Fedora would be less suited for this use case and will not be considered further. Interactions with iRODS correspond to the description in Chp. 3.1.2 on p. 20 (Referenced Content).

3.2.2 Interaction via Fedora (Referenced Content)

Access via Fedora serves the discovery of objects and data export. For export, that is, access to iRODS objects, the system employs the iRODS Davis WebDAV interface. The datastream location is saved together with each datastream and contains the Davis URL allowing objects to be accessed over HTTP.

Authentication over the WebDAV interface is accomplished via iRODS authentication (iRODS login and password).

4 Technical Implementation

4.1 Programming languages

The following section offers information on the programming languages and related technologies employed in this project to make it easier for you to decide if these can fit into and be integrated with your existing applications, how to carry out customizations and whether the necessary know-how for implementation is available.

- iRODS
    - C for microservices, i.e. for internal routines which determine the functionality of the server
    - Jargon API for interacting with the iRODS system
    - Microservices can be written in Python; Python API for interacting with the iRODS system
    - Proprietary rule language, text based collection of rule definitions
- Fedora
    - Java based server
    - REST and SOAP based web services for interacting with the system. Available APIs:
        - API M - management API
        - API A - access API
- Davis WebDAV
    - Java based web application
- `wissgridservice` Module
    - Java based SOAP web service
- GSearch
    - Java based REST and SOAP web services
- Pazpar2
    - C based application
- Basic OAI- PMH Provider
    - Java based Fedora server plugin
- ORE Provider
    - Java based web application

4.2 Access Control (AuthN/AuthZ)

- iRODS
  - User/Password
  - GSI[29]
  - Kerberos (in development)
  - ACLs for authorization
- Fedora
  - User/Password
  - Based on Java Authentication and Authorization Service (JAAS)
  - Fedora Security Layer (FeSL) for authentication and authorization (the latter is still under development)
  - XACML policy based mechanism for authorization
- Davis WebDAV
  - Employs the iRODS mechanisms

At present only admin and robot users based on login/password authentication are supported. A potential goal of further development is matching support of the security features offered by Fedora and iRODS. This could be achieved either by a shared or a separate, synchronized rights and user management system. An alternative to this would be the integration of GSI (which is already supported by iRODS) with Fedora. Using GSI would also address the issue of how best to synchronize rights and user management.

4.3 Log files

- iRODS
  - WissGrid specific: `$IRODS_HOME/server/bin/wissgrid.log`
  - general: `$IRODS_HOME/server/log/rodsLog.<yyyy>.<mm>.<dd>`
  - received messages: `$IRODS_HOME/server/bin/RECV.log`
  - sent messages: `$IRODS_HOME/server/bin/SENT.log`
- Fedora
  - general: `$FEDORA_HOME/server/logs/fedora.log`
- Tomcat: for Fedora, Solr/GSearch[30] and `wissgridservice`
  - general: `$CATALINA_HOME/logs/catalina.out`
- Apache HTTP Server (for Pazpar2 and Proxy mode)

[29] Grid Security Infrastructure
[30] It's also possible to create a separate log file

- dependent on system and web server defaults. Typically in `/var/log/apache2/error.log`

4.4 Performance and scalability

By interacting directly with the iRODS system (i.e. by means of iCommand command line tools) data transfers are optimized automatically by distributing data packets in parallel over multiple threads and socket connections.

One possible avenue for improving performance further in Fedora is setting up a server cluster. However this is currently not a target for development and will thus not be discussed further.

The connection between the Fedora and iRODS systems and in particular their synchronization might be regarded as a potential performance bottleneck. However performance tests using the iRODS-Pound benchmark in which various quantities and types of data objects are transferred to iRODS have shown that this only has a marginal effect on ingest performance (see Chp 5.1).

5 Installation, Customization and Use

Actual details of the installation are listed in the `README.pdf` document (see Appendix A, pg. 40). The following provides a quick overview of the installation process.

In the case of a test environment we recommend installing all components (iRODS, Fedora…) on a host. However, it is also possible and sensible to prefer a distributed set-up. The local installation will proceed as described in `README.pdf`. For a distributed installation the installer needs to run on all hosts but only those install scripts that are relevant to the host are run. Please also note the customizations for distributed environments described in Chp 5.5.

The command `java –jar WissGrid_Deliverable3.5v0.5.0.jar` initiates the installation. The installation tool writes the packages necessary for the repository into a (user prompted) temporary directory from which the installation of the individual components then proceeds.

Important: Because of dependencies between the installed packets it is crucial that you follow the installation order described in the `README.pdf` document. For example, in order for iRODS to be built correctly, `wissgridservice` and its runtime environment (Tomcat) must already be installed and running, since iRODS generates source code (Stubs) during compilation which is needed for the interaction with the service module[31] which in turn is only possible on a running system.

5.1 Tomcat Configuration

In order to avoid an `OutofMemory Error: PermGen Space` error in Solr, allocate sufficient Heap-space to the Tomcat instance hosting Solr.

This is accomplished by starting Tomcat with the following setting in the environment variable `JAVA_OPTS` (`Xms` sets the minimum i.e. start-up and `Xms` the maximum size of the heap):

```
JAVA_OPTS="-server -Xms768M -Xmx1024M -XX:MaxPermSize=768M"
```

If only very large files are being ingested and indexed, i.e. if Solr continues requiring more memory, without more memory available for release, the ingest will fail and Tomcat may not be able to continue normal operation. To avoid this, one needs be aware of the largest expected transfers so that these can be trialed in a test environment.

5.2 iRODS installation

The iRODS installation is divided into the installation of the iRODS server (see `README.pdf` , step 4.2.2) and its customization for WissGrid (see `README.pdf` , step 4.2.4). The first step may be set aside if one wishes to re-use an already existing iRODS server.

[31] The source code is generated on the basis of the Web Service description (WSDL) of the `wissgridservice` module

Following a successful installation, the iRODS server can be started in a terminal window via `irodsctl start`. For more information on the next steps, see 5.5 page 31.

5.3 Fedora installation

The Fedora installer `fcrepo-installer_WissGrid-AP3-Deliverable3.5v0.5.0.jar` is an executable Java archive. Individual steps in the installation can be followed in the `README.pdf` file.

5.4 Installation of other components

Installing the `wissgridservice` module

The installation is prepared by placing the necessary WAR archive (.war file) in the Tomcat deployment server (`$CATALINA_HOME/webapps/`). Customizing the property file `$CATALINA_HOME/webapps/wissgridservices/WEB-INF/wissgridservices.properties` determines, for example, whether versioning should be activated, which iRODS directories should be associated with which object types (cf. Object Types in 2.6, p. 16), if Solr should be used for indexing, under which URLs services such as Davis WebDAV, Solr or Fedora should be accessible etc.

gSOAP Installation

gSOAP is installed as part of the iRODS installation. For further information, please refer to the `README.pdf` document or the section on "Web Services as Micro Services" on the iRODS website.

Davis WebDAV installation

Davis WebDAV is delivered pre-installed in a Jetty servlet container. The installation consists in essence of setting environment variables, the preparation of a certificate and the customizing a configuration file. For further information on the installation please refer to the source code.

GSearch, Pazpar2, OAI-Provider and ORE-Provider installation

All of these applications (with the exception of Pazpar2) need to be integrated as WAR files in Apache Tomcat. For further details on individual applications, please see the next section.

5.5 Customization options

The following section sets out a number of customization options. These should be noted if you choose to follow a non-standard installation, for example, in the case of a distributed server environment; not using `rods` as iRODS admin user; or not using `fedoraAdmin` as the Fedora admin user.

5.5.1 wissgridservice Components

`$CATALINA_HOME/webapps/wissgridservices/WEB-INF/wissgridservices.properties` offers a number of customization properties. Individual attributes are briefly described below. In a distributed installation, the following attributes in particular need to be customized: `IRODS_<...>`, `DAVISURL`, `SOLR_URL`, `FEDORA_URL` and `FEDORA_WSDL_URL`.

- `RECURSIVE_OBJECT_DIRS`: A semicolon separated list of iRODS directories. Data objects placed in these directories need to be recursive object types (see 2.6 on p. 16) and mapped appropriately.
- `SINGLE_OBJECT_DIRS`: As before, however the objects need to belong to the single object type.
- `MULTI_OBJECT_DIRS`: As before, however the objects need to belong to the multi object type.
- `BAGIT_DESCRIPTION_FILE`: Defines the end of a BagIt metadata file and sets the associated naming convention.
- `METS_DESCRIPTION_FILE`: As above, but for a METS metadata file.
- `OAI_ORE_DESCRIPTION_FILE`: As above, but for a OAI metadata file.
- `IRODS_<...>`: Sets the iRODS connection parameters.
- `ZONE`: Sets the iRODS zone.
- `OBJECT_STORE`: Sets the iRODS path for the Fedora object store. This is the location where Fedora saves FOXMLs as managed content (via the low-level storage module).
- `DATASTREAM_STORE`: As above, but for datastreams.
- `DAVISURL`: The URL at which Davis WebDAV can be accessed.
- `FEDORASCHEMA`: Sets the Fedora schema which is used as a prefix for Fedora PIDs or association predicates. Fedora also refers to this as a Fedora namespace.
- `AVU_ATTR_NAME_<...>`: Sets various AVU attributes.
- `AVU_ATTR_VALUE_<...>`: Sets various AVU values.
- `IRODS_VERSIONING`: Specifies whether versioning will be used. This property can only set globally.
- `VERSIONSUBPATH`: Directory used for versions.

- VERSION_MARK: Sets the suffix which distinguishes a version of a data object. The suffix is added system-wide to the version number.
- SOLR_INDEXING: Specifies whether Solr will be used in iRODS
- SOLR_URL: The URL path to the Solr sever
- FEDORA_FORCE: Specifies (on Fedora) if deletion will be forced even if this will cause dependencies to break.
- FEDORA_URL: The URL path to the Fedora server
- FEDORA_WSDL: The URL path to the WSDL Service API-M.

If the URL paths or access information for Fedora or iRODS components change during use then these changes need to be entered into the configuration file and the `wissgridservice` web application has to be restarted.

5.5.2 iRODS

The central facility for customizing the iRODS system are the rule definitions. Changes to rule definitions take effect immediately. For WissGrid, the following two rule definitions in `server/config/reConfigs/` are of interest: `wissgrid.re` (new rule syntax) and `wissgrid.irb` (old rule syntax). If one, for example, wishes to change the email address of the iRODS admin, then the change needs to be made the end of both files. Similarly, if one wishes to change the default resource, the change needs to be made at the end of both configuration files.

If the `wissgridservice` component is not installed locally on the iRODS server (which however, is the recommended practice, see listing of system components in 3.1 on p. 19) then the `wissgridservice`-URL the file `<installation-path>/res/irodssChangeset/do.sh` (i.e. the URL: `http://localhost:8080/wissgridservices/apim?wsdl`) needs to be customized prior to compilation of the iRODS servers. This needs to be undertaken prior to compilation since the code generated for communication with the `wissgridservice` component is dependent on it and would otherwise fail. Similarly, if the URL of the `wissgridservice` component were to change in use the change would need to recorded in the configuration file and iRODS restarted.

5.5.3 Fedora

If Fedora is not installed on the same host as iRODS or the default connection parameters are changed the connection and (admin) user data information needs to be updated in the `objectstoreIrodsaccount` and `datastreamstoreIrodsAccount` beans in the configuration file `<FEDORA_HOME>/server/config/akubra-llstore.xml`. If this data is changed during use the Fedora web application will need to be restarted after the changes are written to the configuration file. To maintain the integrity of data between Fedora and

iRODS it is not permitted to change the iRODS zone of a running system (i.e. it must retain the same iCAT).

5.6 System startup

When the system is started for the first time it is important that iRODS starts before Fedora (i.e. Tomcat) since Fedora stores its standard components (e.g. default content model or default service definition) at startup. If the iRODS server is not already running the Fedora startup will fail.

iRODS

The server can be started, stopped or restarted as follows:

```
$IRODS_HOME/irodsctl {start | stop | restart}
```

David WebDAV

The server can be started, stopped or restarted as follows:

```
/opt/davis/davis/bin/jetty.sh {start | stop | restart}
```

Fedora, Solr and `wissgridservice`

These applications can be starting or stopping the Tomcat server in which they reside, as follows:

```
$CATALINA_HOME/bin/{startup.sh | shutdown.sh}
```

5.7 First steps during use

This section offers an introductory guide to the use of the system and its interfaces.

Fedora:

Fedora can be explored via its web interfaces, the admin console or from the command line.

Web Interfaces[32]

- `http://localhost/fedora`
- `http://localhost/fedora/admin`

Admin console

- `$FEDORA_HOME/client/bin/fedora-admin.sh`

[32] The URL can vary depending on the context path set during installation and the configuration of the Apache HTTP Server proxy module

Command line (Rest-API)[33]

- Create a new object via:

```
curl -request POST "http://user:password@localhost/
      fedora/objects/test:1?label=testwas"
```

- Load a datastream:

```
curl -request GET "http://user:password@localhost/
      fedora/objects/test:1/datastreams/MyDS/content"
            -o my.pdf
```

iRODS iCommands

The iRODS system can be tested via the iCommand command line tools. These are found in `the $IRODS_HOME/clients/icommands/bin/` directory.

- Transfer data objects (ingest): `iput einFile.txt a.txt`
- Create a collection / directories: `mkdir a`
- List objects in the current directory: `ils`
- Help: `iput -h`

Relative paths always refer to `irodsCwd` (current working directory) which was defined in the `irodsEnv` file. The file is created `in ~/.irods/.irodsEnv` following installation.

Davis WebDAV

The Davis WebDAV module can be tested e.g. in the browser.

```
<host>:<port>/
<host>:<port>/<zone>/home/<user>/
```

GSearch

Accessible after installation via: `http://localhost/gsearch/rest` [40]

Pazpar2

Accessible via: `http://localhost/pazpar2/search.pz2` [40]

[33] Also dependent on the context path

5.8 Migration

If you would like to reuse an existing iRODS installation then it is likely that all data in Fedora needs to be re-registered - this process is supported by the rules in the file `<installation path>/res/irods/subsequentFedoraIngest.r`
Re-registration is also of relevance for iRODS paths to which an association between directory path and object type is only established later in the property file (see p. 17).

The subsequent registration (manually initiated callback) is carried out with the following command:

```
irule –F subsequentFedoraIngest.r
      "'/tempZone/home/rods/testColl'" "null"
```

The command is passed to the referenced iRODS collection (with a trailing "/") as a parameter and starts the ingests. If the collection is not found one of the path lists pointing to an object type (`wissgridservices.properties`) either directly or as a sub-path, no registration (no ingest) takes place. The execution of the command is postponed which supports the server in scheduling tasks. The status of the command can be tracked with `iqstat` or `iqstat <id>`.

6 Version Information

This section provides information on system requirements and the modifications carried out on Fedora and iRODS for WissGrid. In addition, it lists changes from previous versions and lists further areas for development and potential weaknesses.

6.1 Minimum System Requirements

At this time iRODS is only supported on Linux and OSX. In addition Fedora requires the Java Runtime Environment (JRE) v1.6 and Tomcat 6.x since Fedora does not (at time of writing) start reliably under Tomcat 7.x. Tomcat also requires a sufficiently large memory heap (see 5.1 on p. 58).

6.2 Modifications to Fedora and iRODS

- During installation the iRODS server (the choice is given between v2.5 or 3.0) is initially installed unchanged and subsequently adapted for WissGrid. This is controlled via an installation script. During modification new microservices and custom rules, as well as configuration are added to the iRODS server. In addition, as part of this process, the script installs gSOAP in order to offer microservices as web-service clients.

- The file `WissGrid-D3.5.3-grid-repository-fedora-irods-installer.jar` (Fedora Installer) includes (in place of the default Akubra-FS plugin filesystem) the iRODS Akubra plugin.

6.3 Differences to version v0.40

- Improved configuration possibilities (`wissgridservice.properties`)
- Support for all three object-types
- Full support for versioning of all object-types
- Interfaces: SRW/SRU, OAI-ORE, OAI-PMH, distributed search
- Use of Apache HTTP server and Apache proxy module

6.4 Potential areas for development

Relevant areas for further development might include:

- Improved support for AuthN/AuthZ
- Modularization, i.e. the conversion to OSGi and the resultant possibility for configuration and replacement of system components (i.e. bundles or modules) etc. during live operation. Work in this area is already underway.

6.5 Known problems/weaknesses

The following are currently known problems:

- Installation is only possible on Linux and OSX

- At this time, iRODS and Fedora can only be used with robot i.e. admin user accounts and all rights and user management needs to be handled on the application layer.
- The employed version of Solr (v1.4.1) has difficulties indexing PDF files. The problem is likely related to the PDFBox implementation which Solr uses.
- If Tomcat is not allocated sufficient heap space larger ingests (e.g. 10x 500MB simultaneously) will fail. The issue is less with Tomcat than with Solr. But since Solr is dependent on Tomcat resources the solution to the problem lies with Tomcat. How much heap is required depends on individual use will need to be determined by means of a test installation.

7 Developer Information

The information in this section is intended to assist developers in familiarizing themselves with the system and to point out entry points for development.

7.1 Procedure Calls

The callback implementation consist at this time of the following rules, the microservices initiated by them, and the `wissgridservice` module and its methods, to which the microservices delegate specific tasks.

For example, at time of ingest via iRODS, the rule `acPostProcForPut` is triggered, which makes a call to the microservice `msiWissGridCallbackForPutObject`. This in turn calls the `wissgridservice` method `addDS`.

- Rules (iRODS-Rules), which are activated for appropriate actions:

    1. **`acPostProcForPut`** – after insertion of a data object
    2. `acPostProcForCopy` – after copying a data object
    3. `acDataDeletePolicy` – before deleting a data object
    4. `acPostProcForCollCreate` – after creating a collection (iRODS directory)
    5. `acPreprocForRmColl` – before deleting a collection
    6. `acPostProcForRmColl` – after deleting a collection

- Microservices (C-Routines), which are called by the previously listed rules. The microservices determined the collection path or the object path and individual metadata attributes such as PID or DSID:

    1. **`msiWissGridCallbackForPutObject`**
    2. `msiWissGridCallbackForCopyObject`
    3. `msiWissGridCallbackForDeleteObject`
    4. `msiWissGridCallbackForCollCreate`
    5. `msiWissGridCallbackForRmColl`

- Service calls employed by microservices (`wissgridservice`):

    1. **`addDS`** – add a new datastream, index and version
    2. `purgeDS` – remove a datastream, update index and delete version)
    3. `ingestFDO` – create a Fedora digital object
    4. `purgeFDO` – delete a Fedora digital object
    5. `addRS` – create a relationship between digital objects
    6. `purgeRS` – delete relationship

7.2 Entry points

7.2.1 Callback

The previously listed rules, microservices and `wissgridservice` methods are ideal entry points to understand a callback's procedural calls. Rules following the new rule language[34] syntax are declared in the file `$IRODS_HOME/server/config/reConfig/wissgrid.re` those following the old language syntax in the file `wissgrid.irb`.

The directory `$IRODS_HOME/modules/webservices/microservices/src/wissgrid` contains the WissGrid specific microservices.

The source code for the wissgridservice module is in the associated web archive. The class `de.unigoettingen.sub.wissgridservices.callback.impl.WissGridAPIMImpl.java` declares the relevant service methods.

[34] iRODS release v3.0 introduced a new rule language with significantly clearer language syntax.

8 Licenses

Development was carried out under the Apache v2.0 License:

```
Copyright 2012 WissGrid-Project

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

              http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
```

The licenses of libraries and other software included in the installer are listed below:

- Fedora Commons Repository: Apache v2.0
  `http://www.fedora-commons.org/software/licenses`
- iRODS: BSD
  `https://www.irods.org/index.php/License`
- Jargon-API: BSD
  `https://www.irods.org/index.php/Jargon_License`
- gSOAP: GPL (GNU Public License)