USER GROUP MEETING
2025 PROCEEDINGS

PUBLISHED BY THE iRODS CONSORTIUM

iRODS
User Group Meeting 2025
Proceedings

17TH ANNUAL CONFERENCE SUMMARY

The iRODS User Group Meeting of 2025 gathered together iRODS users, Consortium members, and staff to discuss iRODS-enabled applications and discoveries, technologies developed around iRODS, and future development and sustainability of iRODS and the iRODS Consortium.

The in-person and virtual four-day event was held from June 17th to 20th, hosted by the iRODS Consortium in Durham, NC with 81 people attending from 12 countries. Attendees and presenters represented over 30 academic, governmental, and commercial institutions.

TALKS AND PAPERS

LIGHTNING TALKS

irods2dataverse: Python package to deposit an iRODS dataset to Dataverse

Danai Kafetzaki – KU Leuven

iBridges shell: Fast and extensible

Raoul Schram – Utrecht University

FriGO: Long term archiving with iRODS in action

Ingrid Barcena Roig – KU Leuven

Core Facilities and You

John D. Martin III – Research Data Management Core at UNC-Chapel Hill

RDM Tech Dutch Community Announcement

Alice Stuart-Lee – SURF

Managing dataflows in a research hospital

Jan de Graaf, Marjolijn Mertz
Netherlands Cancer Institute (NKI)

ABSTRACT

At the Antoni van Leeuwenhoek hospital and Netherlands Cancer Institute (NKI-AVL) the demands for data storage and data management are rapidly evolving. Departments in our institute increasingly integrate their data acquisition and analysis, sparking interdisciplinary research projects. Furthermore, national and international regulations require researchers to make their data FAIR (Findable, Accessible, Interoperable, Reusable). Also, the development of the "-omic" techniques, such as genomic and proteomics, massively increases the size of acquired data. After analysis, this data should be archived for longer periods of time (>10y). Storing this data on rapid and available storage is a waste of resources and money. All these developments necessitate meta-data driven data management.

We have recently deployed iRODS at our institute to facilitate this type of data management. Additionally, we have deployed ManGO as a user front-end to enable easy data access and allow users to search based on meta-data. With this setup we stimulate project-based research, where access is no longer based on departments or groups but on membership of projects. The abstraction of storage resources by iRODS allows seamless integration with Azure, reducing the demands on on-site storage and greatly lowering costs of archival storage for our institute.

We will present the lessons learned during iRODS deployment. In a hospital with an integrated research department there is a demand for secure dataflows. As a case-study we will present how iRODS is used to support dataflow of clinical pathology data to AI for training of neural networks. We show how this technology impacts clinical and fundamental research and how it allows researchers of our institute to share and access data.

iRODS UGM 2025, June 17-20, 2025, Durham, NC

An iBridges update: How easy can we make it for scientists to use iRODS?

Christine Staiger, Raoul Schram, Maarten Schermer
Utrecht University

ABSTRACT

iRODS is a research data management (RDM) system that provides the backend to support researchers in all steps of the data life cycle.

iRODS allows institutes to implement data management policies, e.g. for archiving and publishing data packages. To guide researchers through those data management policies most iRODS-based RDM systems provide a user-friendly webportal. However, those webportals are limited when it comes down to supporting researchers in their daily work with their data and researchers are usually referred to existing tools to interact with iRODS like the iCommands and various iRODS APIs.

For scientists, these tools are not straightforward to use and the step from working with a webportal to working with those powerful tools is often too large. Hence, despite all the effort of building RDM systems based on iRODS, this lack of understandable and accessible tooling means that the full capabilities of iRODS remain unused.

To do their daily work researchers often fall back to storage services which are easier to steer e.g. through WebDav mounts or services which offer sharing of data through shareable links.

iBridges is an open-source project which addresses the above-mentioned problem by developing software packages and tools to actively work with the data in the existing iRODS-based RDM systems. The tools combine user-friendliness with all features iRODS has to offer (from simple data transfer to metadata manipulation). It can be used to work with data on all iRODS-based RDM platforms and addresses scientists of all backgrounds. iBridges provides:

1. a python package for scientific programmers to integrate their compute pipelines with research data management;
2. a command line interface for data transfers for data managers and scientists employing scripting languages (R, Matlab etc.);
3. a generic graphical interface to iRODS targeting the non-programming scientists.

In this presentation we will demonstrate how iBridges tries to improve the end-user experience:

- The onboarding process by presetting and checking environment variables in the environment json file;
- Handling access to different iRODS instances by aliasing and password caching;
- Defining upload and download iRODS paths through a new IrodsPath class which adds python pathlib-like functionality;
- Easing the use of the python-irodsclient's metadata functionality;
- Defining easy-to-use upload, download and synchronisation functionality which also includes the metadata.

iRODS UGM 2025, June 17-20, 2025, Durham, NC

Integrating iRODS into scientific workflows

Raoul D. Schram, Maarten D. Schermer, Matty S. Vermet
Utrecht University

ABSTRACT

Commonly, in a researcher's workflow the raw data resides in an iRODS system, researchers download this data, perform local analysis, and upload the output of this analysis back to this same iRODS system. For more complex analyses this process might happen multiple times. In such cases, the advantage of using standardized workflow methods is reproducibility of analyses, and the possibility of sharing results, workflows, and visualizations with the community.

This can technically be achieved using shell scripting in combination with iRODS command line tools such as GoCommands, iCommands, or iBridges. Workflow frameworks can make the process easier and improve the reproducibility of researchers' work. In the scientific domain, Galaxy, nextflow, and Snakemake are the most widely used workflow frameworks.

Galaxy is an open-source workflow-system, accessed through an interactive, web-based GUI. To create workflows, researchers can drop, drag, and connect the various steps in their pipeline. This makes Galaxy easy to use for researchers with little or no experience in programming.

Before our work there were several existing ways in which Galaxy can interact with iRODS. However, none of these allow end users to configure their credentials and server-specific settings. This effectively forces the use of a single set of iRODS-credentials for all users of a specific Galaxy deployment. Furthermore, implementation prohibits easy integration of these tools in actual workflows.

We have created a new tool that enables researchers to directly use data on iRODS servers in their workflows. Up- and download of both files (data objects) and collections to and from iRODS can be seamlessly integrated into Galaxy workflows. Furthermore, our tool allows for use with individual iRODS accounts and configurations within the same Galaxy instance, rather than using a single set of credentials for every user. Additionally, we developed an interactive Galaxy tool for browsing an iRODS instance, allowing users to explore the filesystems hierarchy and select relevant iRODS paths within the Galaxy environment.

Snakemake is a command line-based workflow management system aimed at creating reproducible and scalable data analyses. An iRODS plugin based on the PRC already exists for SnakeMake. We have adapted this plugin to work on top of iBridges, which greatly simplifies the plugin, adds a few features such as wildcard expansion, and integrates with the iBridges CLI.

We will give a live demonstration of the Galaxy and Snakemake plugins.

iRODS UGM 2025, June 17-20, 2025, Durham, NC

ManGO Platform updates: ManGO Portal, ManGO Ingest, and ManGO Flow

Paul Borgermans, Mariana Montes, Ingrid Barcena Roig, Danai Kafetzaki, Jef Scheepers, Joachim Bovin, Mustafa Dikmen, Ronny Moreas, Peter Verraedt
KU Leuven

ABSTRACT

ManGO is the Active Research Data Management Platform built upon iRODS, offered by KU Leuven to all its researchers. Around it, we have developed a number of modular, open source, Python-based products to address particular needs. ManGO ingest is a standalone ingestion tool that monitors file systems for automatic uploads to iRODS with robust retry mechanisms, enriched with automated metadata extraction, filtering and other features. ManGO Flow, based on the Celery framework, supports refined post-ingest workflows for data management, from filename validation and normalization to metadata extraction to automatic user and group account management and permissions. In addition, it is the framework we use for long running asynchronous operations. Finally, the already known highly customizable ManGO Portal keeps evolving with new developments in reporting and user interactions.

iRODS UGM 2025, June 17-20, 2025, Durham, NC

Enhancing iRODS Monitoring

Alice Stuart-Lee, Francisco Morales
SURF

ABSTRACT

As the data management team of SURF, we support universities and research institutes in the Netherlands by hosting and managing iRODS instances. With the growing adoption of iRODS across these organisations, the need for robust monitoring solutions has become increasingly important. In this presentation, we will share how we've developed a comprehensive monitoring setup that provides insights into iRODS usage and performance.

We'll walk through our monitoring stack, from backend tools (OpenSearch and Logstash) to the front-end dashboards and alerting setup in Grafana. In addition to showcasing the outputs, we will demonstrate the custom automation tools we've built to streamline the creation and customisation of monitoring setups for each client.

A key focus of our approach is the use of iRODS data integrity metrics, which enable us to quickly identify issues such as missing checksums or registered data objects without corresponding physical files. These metrics, combined with usage and availability patterns, have significantly improved our troubleshooting efficiency, saving time that would otherwise be spent combing through logs.

The goal of this presentation is to share our experiences and foster a discussion about monitoring strategies with the iRODS community. By learning from each other's approaches, we hope to help others optimise their iRODS environments and tackle common challenges more effectively.

iRODS UGM 2025, June 17-20, 2025, Durham, NC

Python iRODS Client v3.1.1

Daniel Moore
iRODS Consortium

ABSTRACT

This talk will cover the four releases since last year. This includes many small bug fixes, removal of Python 2 compatibility, and usage of the new authentication framework.

iRODS UGM 2025, June 17-20, 2025, Durham, NC

A data mesh for research data management

Claudio Cacciari
SURF
Moreelsepark 48,
3511 EP Utrecht,
Netherlands
claudio.cacciari@surf.nl

ABSTRACT

In our experience, as data management team of SURF (the Dutch national IT cooperative for education and research), we have seen iRODS adopted successfully in various organizations, like universities or medical centers, or part of them, like departments or laboratories. It is typically used as the "brain" of a data infrastructure to implement storage virtualization, storage tiering and, in general, the full data lifecycle management. However, when we look at research projects that encompass multiple organizations, or multiple units within big, distributed organizations, we notice that there are more difficulties, both of human and technical nature. Sometimes just opening ports in a firewall is quite hard, or, in other cases, the data governance is not well defined or understood. In the last two decades, in that part of the data management field more business oriented, new paradigms and concepts emerged, like data warehouse, data lake and, more recently, data fabric, data product and data mesh. In this presentation we borrow some of those concepts and we map them to research data infrastructures and iRODS to propose a solution that addresses some of the issues of building data platforms for large, distributed infrastructures. We will describe an initial technical and organizational implementation with a perspective for the next steps.

Keywords

Data Mesh, Data Product, FAIR Data, Neptune API, Research Data Management

INTRODUCTION

In our experience, as data management team of SURF (the Dutch national IT cooperative for education and research), we have seen iRODS adopted successfully in various organizations, like universities or medical centers, or part of them, like departments or laboratories. It is typically used as the "brain" of a data infrastructure to implement storage virtualization, storage tiering and, in general, the full data lifecycle management. However, when we look at research projects that encompass multiple organizations, or multiple units within big, distributed organizations, we notice that there are more difficulties, both of human and technical nature. Sometimes just opening ports in a firewall is quite hard, or, in other cases, the data governance is not well defined or understood. How can we make the creation of project or community-oriented data infrastructures easier? How can we make them benefit from existing iRODS and YODA services already well established within partner organizations?

RESEARCH DATA MESH

The reproducibility of research data was and still is a problem, named the reproducibility crisis. The concept of FAIR data was created to address it, providing guidelines to improve Findability, Accessibility, Interoperability, and Reuse of digital assets [1], or, in other words, to improve the quality of the research data.

Findability, accessibility and interoperability of research data have been improved over the past years, through various technologies; however, there are still important gaps in the data infrastructures. Often, they require a level of technical expertise which is not common among the researchers and there is a lack of interoperability among the tools. The result is that the users are forced to create data workflows oriented by the technology or the service which they use, rather than the domain which they belong to.

In the field of business data, a similar problem was described a few years ago [2]. The existing data architectures, Data Warehouse and Data Lake, were considered in a state of crisis, because they were not up to the growing complexity of the data pipelines and of the organizational needs. Therefore, a paradigm shift was proposed to take up the new challenges. It was called Data Mesh and its main purpose is to create a decentralized data architecture that enables the extraction of large-scale analytical data [3].

We propose here to adapt the Data Mesh concept for the field of research data, where its scope can be re-defined in this way: to create a decentralized data architecture that enables the exchange of large-scale FAIR data.

There are four underpinning principles that any data mesh implementation embodies to achieve the promise of scale, while delivering quality and integrity guarantees needed to make data usable:

1. *domain-oriented decentralized data ownership and architecture,*
2. *data as a product,*
3. *self-serve data infrastructure as a platform, and*
4. *federated computational governance.*

Where "federated computational governance" means *a governance model that embraces decentralization and domain self-sovereignty, interoperability through global standardization, a dynamic topology and most importantly automated execution of decisions by the platform* [2].

The first point helps us to address the problem of the fragmentation of data infrastructures among different technologies and services. The idea is to support a different way to provide and consume the data, aggregating them per research domain, community, project.

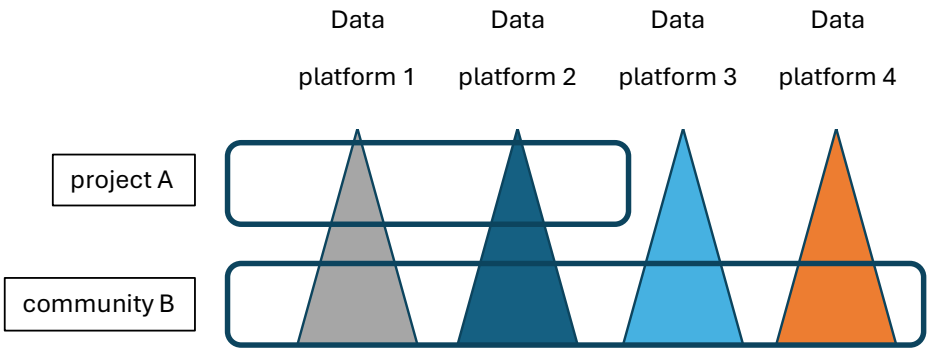

Figure 1. Providing data per domain or project.

In this way, a researcher of project A can focus on working on the project's data, not on data platform 1 and 2. The federated approach breaks silos, which are built around an organization, or a part of it, or around functions of the data life cycle. For example, certain data services provide sharing capabilities, but they are not suitable for long term preservation or viceversa.

A data product is defined by some principles and, as someone noticed [4][5], they are closely related to FAIR data principles.

FAIR	Data mesh characteristics	Why it is important
Findable	Discoverable, addressable	Data product teams can find the data products from other teams quickly and independently
Accessible	Addressable, self-describing, secure, natively accessible	Data product teams can consume the data products from other teams without significant friction or a lengthy process
Interoperable	Trustworthy, secure, self-describing, interoperable, addressable	Data products can be consistently and reliably combined together, they follow standardization and harmonization rules, leading to greater usage
Reusable	Trustworthy, secure, self-describing, interoperable	Data products can serve more than one use case, and are re-used in a new setting creating more value
	Valuable on its own	A data product should carry a dataset that is valuable and meaningful on its own-without being joined and correlated with other data products

Figure 2. FAIR data as Data Products [5].

In the business field, the minimal size of a unit of a data mesh is the data product, so a node in the mesh is a data product. In the research field, we may have similar scenarios, but we have certainly also cases where the Data Mesh node is a set of data products, aggregated by project or scientific domain. An interesting example of such a node is that of a data common, *a cloud-based data platform with a governance structure that allows a community to manage, analyze and share its data* [6].

The third principle, being a self-serve data infrastructure as a platform, gives us two hints. The word "self-serve" indicates the importance of the data consumers. A Data Mesh democratizes the data giving more power to the researchers in discovering and accessing the data without the intermediation of a central team. While the sentence "data infrastructure as a platform" defines the Data Mesh itself as a platform, thus opening the possibility to create a Data Mesh of Data Meshes, where a Data Mesh is a node of a Data Mesh of higher level.

Finally, the last point reminds us that, even if it is a decentralized system, it must have a common governance. The federated approach allows each data owner and provider to maintain control over their own data, dictating the conditions to get access. These conditions have to be harmonized (i.e. not conflicting) anyway with the common Data Mesh rules and automated to support the self-serve capability.

The possibility to use a Data Mesh as a building block for distributed systems has been described also in relation to a Data Space. Data Spaces are *a distributed and standards-based approach to enabling data sharing and use across organizations* [7]. Data governance, authorization and connection capabilities of Data Spaces complement the Data Mesh capabilities, which can be seen as a Data Space in miniature.

IRODS AS A RESEARCH DATA MESH NODE

Given the description of a Research Data Mesh offered in the previous chapter, iRODS and YODA can be seen as nodes of a Research Data Mesh. Then we could offer a solution to the initial problem of the creation of project or community-oriented data infrastructures, building a mesh where the services and the data of the different partner organizations are shared in a FAIR way. The federated governance approach would require an organizational effort to define common rules, but minimal, just enough to define the "borders" of the mesh's space, leaving each partner in control of its own data.

Hub-and-Spoke architecture

Sometime, an organization has multiple services that are involved in the data lifecycle, not just a single iRODS or YODA instance. We could apply the Research Data Mesh paradigm within the organization too. Once connected to the external Research Data Mesh, we would realize a mesh of meshes. However, the more the mesh is stratified in multiple layers, the harder is the challenge of the federated governance. We propose to mitigate that complexity, adopting a hybrid mesh architecture. Where the external Research Data Mesh is coupled with a hub-and-spoke architecture within the organization. A hub-and-spoke design provides a central point, the hub, where it is possible to apply the governance rules and, in general, organization-wide data policies. This central point would represent the whole organization as a node in the external Research Data Mesh, guaranteeing consistency in the exchange of data and metadata between the organization and the other nodes of the mesh.

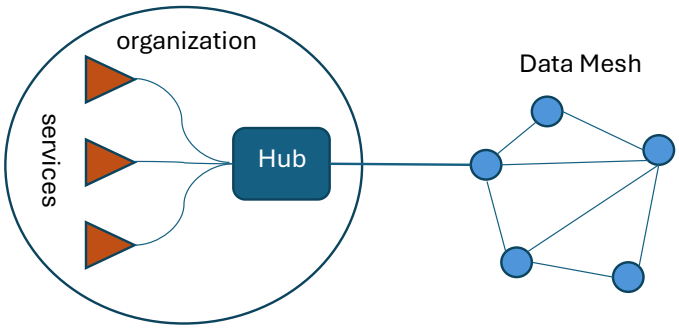

Figure 3. Hybrid data mesh.

However, the hub in the hub-and-spoke architecture could become a bottleneck and a single point of failure. A redundant implementation can mitigate the latter weakness, but in order to remove the former one, we propose limiting the scope of the hub to metadata. In fact, any data workflow can be designed so that it is metadata driven, hence if we keep the metadata consistent, the governance and data policies rules will be consistent as well. On the other hand, the data will be transferred through a direct connection to or from the service hosting them, so that the performance and the scalability will be optimized.

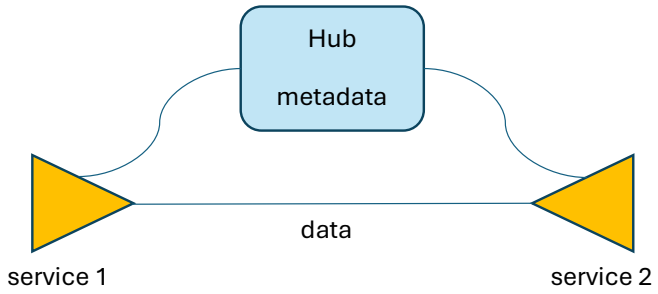

Figure 4. Hub for metadata only.

Research Data Hybrid Mesh node

Which features should offer the node of this Research Data Hybrid Mesh, being at the same time the hub of a hub-and-spoke topology?

We propose that it is defined by the following properties:

1. It is a service registry: it needs to be aware of all the services within the organization
    a. to advertise them to the other nodes of the mesh.
    b. To support the creation of data workflows that use different services.
2. It is a user registry: it needs to identify any "actor" that interacts with it to support decisions based on identity and role.
3. It is a metadata catalog: the metadata related to users, providers and data are necessary to enforce policies based on them, automate as much as possible the decisions and to fulfill the definition of data product.
4. It is an API gateway: it needs to route and control the API requests between the organization domain and the external world.
5. It is a data sharing point: where the available data can be advertised to other users, providers, nodes ... And where data can be discovered through metadata and requested from the data provider hosting them. As explained above, the data should not be transferred through the node, but directly from/to the data provider.

These properties are familiar to those who have some knowledge of iRODS, because they overlap largely with its features. Therefore, in the case that an instance of iRODS, or YODA is the only service connected to the Research Data Mesh, the hub-and-spoke architecture could "collapse" on iRODS itself, while in case of multiple services an additional logical layer, the hub, is needed.

CONCLUSION

We have identified fragmentation, from a technological and organizational point of view, as one of the main obstacles hindering the creation and effectiveness of large data infrastructures. We have adapted a paradigm and an architecture, the Data Mesh, born in the business field, to the research data field and we have shown that it can help in overcome the fragmentation issue, democratizing and simplifying the access to the data. iRODS (and YODA) can be one of the building blocks of such Research Data Mesh. Generalizing the solution for organizations with multiple data services, we can design the node of the mesh as a hub of and-and-spoke architecture, so that multiple data services are connected to the mesh via the hub. In this case, iRODS would be one of the services connected to the hub. To support the proposed solution, the hub should act as:

1. A service registry.
2. A user registry.
3. A metadata catalog.

4. An API gateway.
5. A data sharing point.

REFERENCES

(1) https://www.go-fair.org/fair-principles/ (visited on May 9th 2025)
(2) https://martinfowler.com/articles/data-mesh-principles.html (visited on May 9th 2025)
(3) Inês Araújo Machado, Carlos Costa, Maribel Yasmina Santos,
Data Mesh: Concepts and Principles of a Paradigm Shift in Data Architectures,
https://doi.org/10.1016/j.procs.2021.12.013.
(https://www.sciencedirect.com/science/article/pii/S1877050921022365)
(4) https://www.linkedin.com/pulse/fair-guiding-principles-data-mesh-jeroen-angenent/ (visited on May 9th 2025)
(5) https://www.thoughtworks.com/insights/blog/data-strategy/data-mesh--helping-life-sciences-organizations-get-more-from-the (visited on May 9th 2025)
(6) Grossman R. L. (2023). Ten lessons for data sharing with a data commons. Scientific data, 10(1), 120.
https://doi.org/10.1038/s41597-023-02029-x
(7) Antti Poikola (Sitra), P J Laszkowicz, Ville Takanen and Teemu Toivonen (Futurice) (2023), Technology Landscape of Data Spaces. Sitra Publisher.
(https://www.sitra.fi/en/publications/technology-landscape-of-data-spaces)

Kando: An iRODS Compatible Data Organizer for CKAN

Tanmay Dewangan, Tony Edgin, Nirav Merchant
CyVerse / University of Arizona

ABSTRACT

As scientific data grows in scale and complexity, researchers increasingly rely on disparate infrastructure, like local servers, cloud buckets, and institutional archives, to store and manage data. However, this fragmentation makes it difficult to build a cohesive, discoverable, and FAIR-aligned data commons.

This talk presents Kando, which is a metadata curation tool that connects iRODS-managed data and metadata with a Comprehensive Knowledge Archive Network (CKAN) data commons. iRODS serves as the underlying data infrastructure within CyVerse and is used to manage metadata and files across a 7.2 PB research data store. Kando is designed to work within the CyVerse environment, enabling researchers to extract, standardize, and publish metadata from iRODS using modern standards like DCAT and Croissant. It inventories datasets, collects metadata (such as authors, file size, content type, custom tags, etc.) and transforms these records into searchable, CKAN-compatible datasets that link back to the original files. Kando applies a similar approach to public AWS S3 and Google Cloud Storage buckets by allowing researchers to incorporate cloud hosted project data into a unified CKAN data commons.

By unifying metadata from both institutional and cloud sources, Kando provides an intuitive user interface that makes it easier for researchers to share, discover, and access data through a single CKAN portal. This streamlined approach reduces the time and effort needed to curate datasets, supports FAIR data practices, and helps research communities build flexible, project-specific data commons that promote collaboration and reuse.

iRODS UGM 2025, June 17-20, 2025, Durham, NC

iRODS Build and Packaging: 2025 Update

Markus Kitsinger
iRODS Consortium

ABSTRACT

This talk will, again, provide an update on our journey to 'Normal and Boring' with regard to CMake, libstdc++, and building for various platforms. We're closer than we've ever been.

iRODS UGM 2025, June 17-20, 2025, Durham, NC

iRODS Roadmap 2025

Kory Draughn, Terrell Russell
iRODS Consortium

ABSTRACT

iRODS 5.0 has been released and our 11-year backwards-compatibility promise has been kept. The community's input for the next steps for the iRODS server have been incorporated into the roadmap. This talk will discuss the current plans for the future.

iRODS UGM 2025, June 17-20, 2025, Durham, NC

Efficient data staging with iRODS HTTP API in the LEXIS Platform 2

Marek Nieslanik, Martin Golasowski
IT4Innovations, VSB-TU Ostrava

ABSTRACT

The LEXIS Platform is a solution for easy and secure access to complex computing workflows running on supercomputers or Cloud with distributed data management features based on iRODS. We present the latest updates of the LEXIS Platform along with several use cases focusing on the data management. One of the recent additions is migration to the iRODS 4.3 and its native HTTP API along with support for OpenID. In our talk, we describe the full solution along with the new approach to user space data staging between HPC and iRODS using in-memory buffers, including benchmark results.

iRODS UGM 2025, June 17-20, 2025, Durham, NC

Metadata schemas updates: JSON schemas and storage in iRODS

Mariana Montes, Paul Borgermans, Joachim Bovin, Danai Kafetzaki, Jef Scheepers, Mustafa Dikmen, Ingrid Barcena Roig
KU Leuven

ABSTRACT

At KU Leuven, researchers are encouraged to group, structure, validate, and describe their metadata with metadata schemas. We have developed a tool to create forms that can be used to fill in the metadata in our web portal, and a Python package to validate and convert metadata to namespaced iRODS AVUs from Python dictionaries. The latest developments include storing the schemas in iRODS and describing them as JSON Schema, which would allow implementing iRODS's server-side validation. This also fits our goals of integration and interoperability, as it facilitates importing and exporting even schema metadata from and to JSON for different purposes. In this talk we will introduce these changes and illustrate metadata schema user workflows, particularly in combination with other automated processes.

iRODS UGM 2025, June 17-20, 2025, Durham, NC

FriGO: the KU Leuven long-term archiving solution with iRODS

Mariana Montes, Ingrid Barcena Roig, Mustafa Dikmen, Joachim Bovin, Danai Kafetzaki, Paul Borgermans, Jef Scheepers
KU Leuven

ABSTRACT

In recent years, researchers are increasingly faced with the challenge of properly managing research data in all phases of the data lifecycle, from the start of the project to publication and long-term preservation. In order to support them conducting robust and high-quality research, research institutions must prioritize Research Data Management (RDM), offering central, integrated support services for RDM.

In this context, KU Leuven has made significant investments in RDM infrastructure over the past years, resulting in a solid ecosystem, with an institutional Research Data Repository (RDR) based on Dataverse to facilitate data publication, and the ManGO platform based on iRODS to manage active research data. However, a third important component was missing: a solution for long-term storage of research data that is not to be published but needs to be kept safe, to ensure reproducibility and possible reuse. In order to fill this gap, we have developed FriGO, the KU Leuven long-term archiving solution, also built upon iRODS.

Active collections stored in ManGO can be selected for archiving via the ManGO Portal. Users must then provide dataset-level metadata to ensure that datasets are properly described, including authorship, access rights, and the lifetime established following the KU Leuven RDM policies. Once the metadata is ready the collection can be prepared and packaged: metadata stored in iRODS both at the data object and collection level is exported as sidecar files, the collection is turned into a BagIt wrapped in an RO-Crate, and the final package is tarred. Upon approval by a responsible party of the research project and the support team, the tar is archived and only the metadata is made available to the users, mostly in the form of an RO-Crate Metadata Document.

In this presentation we will present the product, currently in its piloting phase, focusing on its design and architecture.

iRODS UGM 2025, June 17-20, 2025, Durham, NC

irods4j: A new Java client library designed for iRODS 4.3.2+

Kory Draughn
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
korydraughn@renci.org

Terrell Russell
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
unc@terrellrussell.com

ABSTRACT

This new library exposes both a familiar low-level API similar to the existing iRODS C/C++ APIs and an easier-to-use high-level API. This talk will introduce this library, its design, and remaining work.

Keywords

iRODS, java, jargon, irods4j, client library

HISTORY AND MOTIVATION

The iRODS ecosystem has had a long history of client libraries in multiple languages serving to abstract the iRODS wireline protocol, allowing application developers to work in the language of their choosing. The Java community has had access to the Jargon library [1] since the earliest days of Storage Resource Broker (SRB) development at the San Diego Supercomputer Center (SDSC).

Jargon (originally known as JARGON) was first released alongside SRB 2.1 in June 2003 [2] and was primarily written by Lucas Ammon Gilbert. This work was additionally authored and maintained by David R. Nadeau, John Moreland, and Arun Swaran Jagatheesan.

Through 2005, JARGON stood for "Java Api for Real Grids On Networks" (version 1.4.9 [3]). By late 2006, a second-generation JARGON stood for "Java Api for Rules, Grids, and Online Networks" (version 1.4.25 [4]) as it was ported to work with the newly created (and rule-oriented) iRODS project.

The code hosting itself has moved a number of times. The code repository was initially stored in Concurrent Versions System (CVS) but was moved to Subversion, hosted at googlecode.com, in May 2008. The initial individual commits have been lost but the CVS log of changes is still available [5].

In October 2009, the repository was moved again to irodssvn.ucsd.edu by Wayne Schroeder to be alongside the Data Intensive Cyber Environments (DICE) group at the University of California San Diego (UCSD) as part of the release of iRODS 2.3 [6].

The Jargon code was pulled into its own repository by Mike Conway at the Renaissance Computing Institute (RENCI) with the release of iRODS 2.5 in February 2011 [7]. The existing code was moved into a jargon-trunk repository, and a new refactored third-generation version was created as jargon-core.

The repository was moved again to GitHub by March 2014 with the release of iRODS and Jargon 3.3.2. Mike Conway's maintenance of the library ended with the release of 4.3.2.5 in February 2022.

Kory Draughn's first release of Jargon was 4.3.4.0 in August 2024, and we think this month's 4.3.7.0 release of Jargon may be the last.

iRODS UGM 2025 June 17-20, 2025, Durham, NC

With all of that, after 22 years, the Jargon library has become a relic of an earlier time. It is relatively hard both to maintain and to reason about its API. Its design makes it difficult to support certain usage patterns, including connection sharing and reuse. It provides a high-level, abstract interface that results in a lack of developer control and network traffic that is oftentimes chatty and unnecessary.

For a new library (introduced in this paper), we are interested in providing a simpler approach, something low-level that offers developers the features directly available in the C and C++ iRODS APIs. It will be published at the industry-standard Maven Central Repository for ease of discovery and integration with other projects.

This new library is now available as `irods4j` [8].

WHAT DOES THIS MEAN FOR JARGON?

The Jargon library is officially deprecated. Any applications relying on Jargon should plan to migrate to irods4j. New applications should consider building on irods4j. Jargon may be updated, but it will be limited to changes in support of the iRODS Consortium maintenance releases of Metalnx [9].

WHAT DOES irods4j PROVIDE?

The new irods4j project is published to the Maven Central Repository. This allows developers to easily find and integrate irods4j, its minimal dependencies (jackson, junit, log4j, and woodstox), and its build infrastructure.

irods4j offers both a high-level and a low-level API. The high-level API intentionally mirrors the iRODS C++ API while the low-level API intentionally mirrors the iRODS C API. Documentation provided for the iRODS C API doubles as documentation for the low-level interface of irods4j. This dual approach provides an ease of use when desired and more control when necessary.

Additionally, there are two implementations for each release, one for the stable Java 17 community, and one for the prior LTS version, Java 8. We intend to support both until the community demand for Java 8 wanes and/or the complexity of backward compatibility becomes too high.

Both the two levels of API and two versions of Java provide flexibility for the Java developer looking to build against an iRODS infrastructure.

The classic iRODS native authentication scheme and the new pam_password authentication scheme are both supported. Additional schemes may be added later as necessary.

Regarding the networking possibilities available, all socket options are configurable and under the direct control of the developer. Additionally, this library supports SSL/TLS for secure communications with the iRODS server.

DESIGN

The design of this library stems from two primary goals. The library must be easy to maintain over time. The iRODS protocol continues to evolve and this library should stick to the functionalities made available from the server. This is most easily done if the mapping from library to protocol is as direct as possible. The second goal is to provide maximum control to the developer. Again, this is best realized by making little to no additional choices for the developer over what is provided by the protocol and the server functionality.

This library uses the XML protocol to communicate with the iRODS server. As a new library, this was the easiest path to develop and to commit to maintain. Recent work to prove the relative efficiency of the binary protocol has opened a discussion about adding support to this library, but that has not been decided at this time.

The iRODS protocol's packing instructions have been serialized to XML and this library uses the jackson library to handle that work. Mirroring the iRODS C API future-proofs this library and supports the goal of keeping close to

the protocol's offerings.

```
var host = "localhost";
var port = 1247;
var username = "rods";
var zone = "tempZone";
var errInfo = new RErrMsg_PI();
RcComm comm = IRODSApi.rcConnect(
    host, port, username, zone, Optional.empty(),
    Optional.empty(), Optional.empty(), Optional.of(errInfo));
IRODSApi.rcDisconnect(comm);
```

EXAMPLES

The following few sections demonstrate basic ways to interact with an iRODS server via irods4j.

Connecting to a server

`IRODSConnection` provides an easy way for developers to connect to an iRODS server. It manages a single `RcComm` and is inspired by the C++ `irods::client_connection` library.

```
try (var conn = new IRODSConnection()) {
    conn.connect(host, port, new QualifiedUsername(username, zone));
    conn.authenticate("native", password);

     // Do work.
}
```

Connection Pooling

`IRODSConnectionPool` manages a pool of connections.

Developers can use this with `rcSwitchUser` to write client-side server applications similar to the iRODS HTTP API [10].

```
// Create a pool containing 10 connections.
try (var pool = new IRODSConnectionPool(10)) {
    // Authenticate each connection in the pool.
    pool.start(host, port, new QualifiedUsername(username, zone), conn -> {
        try {
            IRODSApi.rcAuthenticateClient(conn, "native", password);
            return true; // Let the pool know authentication was successful.
        } catch (Exception e) {
            return false; // There was an issue, DO NOT use the pool!
        }
    });
    try (var conn = pool.getConnection()) {
        // Do work.
    }
}
```

Listing the contents of a collection

`IRODSCollectionIterator` allows developers to iterate over a collection's contents. `IRODSRecursiveCollectionIterator` can be used to iterate over subcollections.

```
var conn = // Our connection to iRODS.
var collection = // Absolute path to a collection.

for (var e : new IRODSCollectionIterator(conn, collection)) {
    // Print out the logical path of "e".
    System.out.println(e.path());

    // Inspect "e" for information about the collection entry.
}
```

Writing a data object

`IRODSDataObjectOutputStream` gives developers a simple and familiar way to write data to a replica. It is built on top of `IRODSDataObjectStream` and is inspired by the C++ `irods::dstream` library.

```
var conn = // Our connection to iRODS.
var logicalPath = // Absolute path to a data object.
var truncate = true;
var append = false;
var data = "irods4j is easy to use.";

// Create a new data object and write some data to it.
try (var out = new IRODSDataObjectOutputStream(
                    conn, logicalPath, truncate, append)) {
    out.write(data.getBytes(StandardCharsets.UTF_8));
}
```

Reading a data object

`IRODSDataObjectInputStream` gives developers a simple and familiar way to read data from a replica. It is built on top of `IRODSDataObjectStream`.

```
var conn = // Our connection to iRODS.
var logicalPath = // Absolute path to a data object.
var buffer = new byte[512];

try (var in = new IRODSDataObjectInputStream(conn, logicalPath)) {
    // Fill "buffer" with data from the data object.
    in.read(buffer);
}
```

Rule Execution

`IRODSRules` exposes a simple interface for executing rules.

```
var conn = // Our connection to iRODS.
```

```
var name = "irods4j";
var role = "Java client library for iRODS";

// Set the rule text, inputs, outputs, and rule engine plugin instance to use.
var ruleArgs = new RuleArguments();
ruleArgs.ruleText = String.format("*name = '%s'; *role = '%s'", name, role);
ruleArgs.input = Optional.empty();
ruleArgs.output = Optional.of(Arrays.asList("*name", "*role"));
ruleArgs.ruleEnginePluginInstance = Optional.of(
    "irods_rule_engine_plugin-irods_rule_language-instance");

// Execute the rule and print the values of "*name" and "*role".
var results = IRODSRules.executeRule(conn, ruleArgs);
System.out.println(results.get("*name"));
System.out.println(results.get("*role"));
```

FUTURE WORK

This library is new and there have been two releases so far. v0.2.0 is available as of this writing.

There is a lot of work to be done to provide full iRODS API coverage and full testing coverage. We plan to fully document the public API and to leverage GitHub Actions to automate various maintenance tasks.

We look forward to feedback for this fourth-generation library.

REFERENCES

[1] Jargon, iRODS Java Client Library `https://github.com/DICE-UNC/jargon`

[2] SDSC Releases Version 2.1 of the Storage Resource Broker Data Management Middleware, June 19, 2003. `https://www.sdsc.edu/news/2003/PR062003.html`

[3] JARGON 1.4.9 (via Wayback Machine), February 24, 2005 `https://web.archive.org/web/20050301120047/http://www.sdsc.edu/srb/jargon/`

[4] JARGON 1.4.25 (via Wayback Machine), December 3, 2006 `https://web.archive.org/web/20061203051714/http://www.sdsc.edu:80/srb/jargon/`

[5] Moved from CVS to SVN at googlecode.com, May 6, 2008 `https://github.com/irods/irods-legacy/blob/f4f8769f2de5eefae2c254cdd7d01fcdbad6c3ea/iRODS/jargon/jargon.extrods.googlecode.log#L852`

[6] Moved from extrods.googlecode.com to irodssvn.ucsd.edu, October 12, 2009 `https://github.com/irods/irods-legacy/commit/f4f8769f2de5eefae2c254cdd7d01fcdbad6c3ea`

[7] Moved to RENCI, February 2011 `https://github.com/irods/irods-legacy/blob/2.5/iRODS/jargon/RELEASE_NOTES.txt`

[8] irods4j - A Java 17 client library designed for iRODS 4.3.2+. `https://github.com/irods/irods4j`

[9] Metalnx `https://github.com/irods-contrib/metalnx-web`

[10] Draughn, K., Russell, T. (2023) iRODS HTTP API, iRODS User Group Meeting 2023. Durham, NC. `https://irods.org/uploads/2023/Draughn-iRODS-HTTP_API-paper.pdf`

A team approach to enabling streamlined generation, aggregation, management, and reuse of primary data

Rory Macneil
Research Space

Terrell Russell
iRODS Consortium

John D. Martin III
Research Data Management Core
at UNC-Chapel Hill

ABSTRACT

Research institutions face a growing challenge in managing the increasing volume of scientific data while maintaining its long-term value and reusability. Researchers currently struggle with fragmented workflows where data storage, metadata capture, and data publishing are disconnected processes. This leads to inefficient storage usage and increased cost, degraded metadata quality, and ultimately reduces the potential for data reuse. While solutions exist for individual aspects of this challenge (e.g. RSpace for capturing the research process, iRODS for storage management, Dataverse for publishing), they typically operate in isolation or with limited overlap. Applying the principle of Vertical Interoperability, the project presented here proposes to explore approaches to integrating research tools belonging to different stages of the instrument data lifecycle into a seamless workflow environment for instrument data that simultaneously serves researchers' immediate needs, optimizes institutional IT resources, and enhances the institution's research impact through improved data reusability.

Explore an end-to-end solution through the practical development of a prototype workflow to concretely understand pain points and design decisions that are needed from the generation of primary research data by instruments to highly re-usable and discoverable research data

RSpace serves as the daily companion for researchers, acting as an orchestrator of the data flow. It captures experimental context and collects instrument and process/workflow metadata from the point of initial data generation using highly interoperable metadata schema and persistent identifiers for instruments. Data is stored in iRODS providing intelligent data storage infrastructure that enables persistent and controlled access to data objects while efficiently managing storage resources. Data products of the raw data, e.g. obtained through local or high-performance computing are published through RSpace to Dataverse to enable efficient data discovery through federated access and persistent identifiers (DOIs). Original/raw data is linked persistently from Dataverse, while storage and access are managed by iRODS. Other researchers are enabled to find and re-use this data. RSpace, iRODS, and Dataverse will serve as representative tools of their category, and the focus will be on exploring how they can best serve their roles as they're described in this paragraph.

iRODS UGM 2025, June 17-20, 2025, Durham, NC

MetaInx v3.1.0

Justin James
iRODS Consortium

ABSTRACT

This talk will cover the v3.0.0 major release. It removes the need for the PostgreSQL database, improves compatibility with iRODS 4.3, and updates several dependencies.

iRODS UGM 2025, June 17-20, 2025, Durham, NC

Much to learn, you still have: Experiences with Yoda

Sirjan Kaur
Utrecht University

ABSTRACT

Yoda is an iRODS-based research data management solution developed and maintained by Utrecht University to meet research data management challenges. Yoda enables researchers to preserve, share, archive, and publish their research data in compliance with FAIR principles.

Since its launch in 2015, Yoda has significantly improved and expanded to more than 13,000 users and over 4 petabytes of research data. In March 2023, the Yoda Consortium was formed to allow effective collaboration on the development of Yoda, support for researchers, and sharing of knowledge between universities and organizations across the Netherlands. To accommodate this growth and to maintain product quality, we implemented testing strategies for Yoda and presented them at iRODS User Group Meeting 2024.

In this session, we will present an update on these testing strategies, and we will discuss our technical enhancements over the past year, including the upgrade to Python 3 and iRODS 4.3.4, the use of static type checking, and other changes.

iRODS UGM 2025, June 17-20, 2025, Durham, NC

Exploring the iRODS Native protocol, a hidden gem

Ton Smeele
Utrecht University
The Netherlands
a.p.m.smeele@uu.nl

ABSTRACT

The iRODS grid deploys a versatile protocol for client-server communications. While its XML serialization variant is used by many client applications, the more efficient Native variant is currently only available to C-language based clients. We discuss a design for an iRODS client library that also supports the Native protocol variant in languages other than C. We validate the usability of our design via an example implementation in Java. This example implementation is utilized at Utrecht University by a Java application that transfers data objects between unfederated iRODS zones.

Keywords

iRODS, data grid, distributed client server model, communication protocol

INTRODUCTION

iRODS deploys a distributed client/server communication model. Clients send requests to an arbitrary server in the grid. The fulfillment of such a request may require the connected server to consult other servers located in its own grid or in a federated grid. Additionally, when the client so desires, data file content is transmitted efficiently via peer-peer connections directly between the client and the resource server that manages the replica.

All grid communications are based on a single protocol, the *Distributed Shared Collection Communication Protocol*, more commonly known as the iRODS protocol [1]. Its power and flexibility is underpinned by decades of usage, without a need for major change. According to Arcot Rajasekar, the iRODS protocol shares its lineage with the SDSC Storage Resource Broker (SRB), an iRODS predecessor created in 1995 [2]. Parts of the SRB protocol design are based on a request/answer protocol that was deployed in Postgres95 [3]. To accommodate the exchange of more complex data structures as used in SRB and iRODS, significant changes have been applied.

Before a message is exchanged, it is serialized using either *Native* or *XML* encoding. Native is the original protocol implementation, where each element's value is packed as a sequence of bytes [1]. This method is supported by the C/C++ iRODS client library. For instance the iCommands communicate with an iRODS server via the native protocol. Unfortunately, this client library is not readily available to applications written in other programming languages. As a workaround, an XML-based serialization method has been added. Here, elements are serialized as tagged strings. Binary data is base64 encoded. Many iRODS clients depend on the XML protocol, for instance Jargon (Java), PRODS (PHP), PRC (Python), and the Go-iRODSClient (Go) [4][5][6][7].

Messages serialized using the Native protocol are smaller in size compared to messages encoded with the XML protocol. The larger message size produced by XML encoding is caused by a need to encode element names as tag in addition to the element values. In contrast, the Native protocol only encodes the element value. Furthermore, it encodes numerical and binary data more tightly. Despite being more efficient, support for the Native variant has remained limited to C/C++.

iRODS UGM 2025 June 17-20, 2025, Durham, NC

What does it take to implement a client library for the Native protocol? Our study aims to answer this question by creating a prototype of a iRODS client library in Java. We select Java, since it supports object oriented programming with strict type checking. These capabilities facilitate a clean implementation and fast debugging.

CLIENT LIBRARY ARCHITECTURE

We design our client library as a layered architecture, shown in Figure 1. The foundation is a layer that maintains a TCP/IP socket connection between the client application and an iRODS server. The middle layer is responsible for sending request messages and receiving response messages efficiently across this connection. Messages are (de)serialized using either the native or the XML protocol. The top layer implements a client application programming interface (API), that creates a message from a client request, and translates a received message back into a response handed over to the client. Each method present in this layer corresponds to an iRODS server supported API request type. Request arguments are encoded as iRODS data structures. Responses are decoded to Java classes suitable for consumption by client applications.

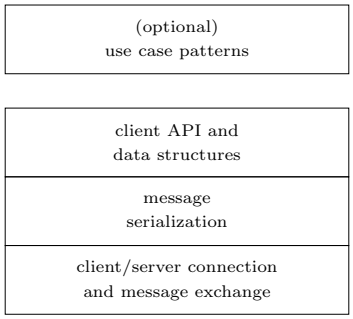

Figure 1. Architecture of MyRODS client library

Some interactions, for instance the transfer of replica data, require the execution of a sequence of multiple iRODS requests. We have developed a set of Java classes as a layer on top of the client library, to support common use case patterns, such as user authentication and parallel data transfer.

Client/server connection and message exchange

For backward compatibility reasons, an iRODS client must always initiate an unencrypted connection. Once connected, the client may request the server to switch to encrypted communications, and vice versa. For instance, this mechanism is used by iRODS to securely communicate a PAM password over an otherwise unencrypted connection. We design encrypted communications as a secure socket layered over the existing socket. The socket switch takes effect after the response has been received and the channels are flushed.

iRODS supports communication between clients and servers with different hardware or software architectures. Regardless of architecture, data is exchanged in big endian format, in compliance with RFC 1700 [8]. Characters are UTF-8 encoded, in compliance with RFC 5198 [9]. To encode data types in big endian, we use the Java class `ByteBuffer`. The method `getBytes` of the class `String` facilitates encoding in UTF-8 character set. When decoding, ByteBuffer assists to meet word alignment requirements of the local architecture.

Virtually all exchanged messages use the layout shown in Figure 2. Each message consists of five components: the header size, header, message, error message, and byte stream. Once the header size, a 32 bit integer, is read, the receiving party knows how to read the header. The header contains data on the sizes of the remaining components, which allows those components to be read in one go. Empty components are allowed, they have a size value of zero.

While the last component is simply a byte array, the header, message, and error message are serialized named iRODS

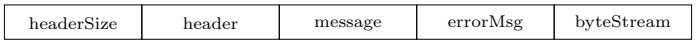

Figure 2. Exchanged message layout

data structure types. The data types of the header and error message are fixed, they are respectively `MsgHeader_PI` and `RError_PI`. The data type of the message component varies by message type.

The iRODS protocol specifies that the header *must* be serialized using the XML protocol. In its first message, the client specifies the desired serialization protocol, Native or XML, for serialization of the message and error message components of subsequent messages. Hence a client library must always be able to support the XML protocol, while support of the Native protocol is optional.

The XML protocol has changed slightly as of iRODS 4.2.9. To fix a confusing specification in the protocol, iRODS now escapes an apostroph using the code `&apos;` whereas in prior releases this code would represent a backtick. The new encoding is used only when both client and server are version 4.2.9 or later. We use the server version, reported in an initial `RODS_VERSION` type response message, to choose the appropriate XML protocol variant.

Message serialization

In our message serialization layer, we deploy a strategy design pattern to implement all of the (de)serialization variants mentioned in the previous section. Serialization is done using the class `Packer` while deserialization is performed by `Unpacker`.

Our implementation must support the (de)serialization of six iRODS base data types: `bin` holds an arbitrary byte value, `char` depicts a byte used as a character, `str` is a null-terminated variable length character array, `int16` and `int` are respectively 16-bit and 32-bit size integers, and `double` represents a 64-bit size floating point value. In practice, the double type is used as a union to store either a floating point or, more often, a long integer value. We design an abstract class `Data`, with subclasses per iRODS data type. Each instance will store a variable name and its value.

We also need to support the serialization of composed data types. iRODS provides `struct` along with pointer reference and array as mechanisms to compose more complex data types from the iRODS base types. These concepts have semantics similar to their C language pendant. iRODS annotates its data type specifications with dimension information, so that the receiving party can allocate an appropriate amount of memory for each data element, for instance in case of indirect references. This information is important for client library implementations in programming languages such as C, where the programmer is responsible for memory management. As memory management is implicit in Java, our implementation may ignore dimensions for data received. However, it will need to ensure that any data sent does not exceed specified dimensions, for example the maximum size of a variable-length character string must be respected. We use a composite design pattern to represent the composition of data types.

The iRODS protocol requires messages to contain instances of predefined struct types, well-known by client and server. This approach allows messages to have a significantly smaller footprint, since details on the structure of data do not need to be included in each message. Note that this design assumes that client and server use a sufficiently compatible set of struct specifications.

```
GenQueryOut_PI = "int rowCnt; int attriCnt; int continueInx; int totalRowCount;
                  struct SqlResult_PI[MAX_SQL_ATTR];"
SqlResult_PI   = "int attriInx; int reslen; str *value(rowCnt)(reslen);"
```

Figure 3. Packing instructions for struct GenQueryOut_PI and a struct it depends on

All predefined struct types are specified using *packing instruction* strings. The packing instruction is a ordered sequence of element names along with their base type and dimension. Figure 3 shows the packing instruction for

the predefined struct `GenQueryOut_PI`. Instances of this struct will consist of four integers, and an array of elements of the struct type `SqlResult_PI`. This example demonstrates several features of the packing instruction syntax. A specification can be nested by referencing another struct, such as `SqlResult_PI`. Instead of specifying a dimension using the literal value `50`, we may reference a constant such as `MAX_SQL_ATTR`. To specify the dimension of a dynamic array, which is only known at runtime, we can refer to the value of another instantiated variable (already in scope). Note for example, how the actual value of `rowCnt`, received at runtime as part of a `GenQueryOut_PI` data structure, will be used to drive the size of the subsequently received pointer array `value` in the embedded struct `SqlResult_PI`. Our implementation deploys a map `PackInstructionsConstants` to store all constants and builds a `Data` structure containing processed variables. Whenever a message is deserialized, the packing instruction of the expected data structure is parsed by class `ParsedInstruction`, searching the map and processed-data structure to resolve references as needed.

```
MsParam_PI     =  "str *label; piStr *type; ?type *inOutStruct; struct *BinBytesBuf_PI;"
```

Figure 4. Packing instruction for MsParam_PI includes a dependent type

In addition to the above mentioned features, packing instructions support a so-called *dependent type*. Here, the type of an embedded struct is determined dynamically at runtime based on the value of an instantiated variable, similar to the mechanism used for dynamic arrays. For this purpose, iRODS has added a meta data type `piStr` to the set of data types. This name is an acronym of *packing instruction string*. Values of this type are variable-length character arrays, restricted to the set of predefined struct names. Figure 4 demonstrates the use of dependent types. The symbol `?` is part of the packing instruction syntax, and indicates that the immediately following `piStr` variable must be dereferenced, to obtain the name of the target struct type. In the example case, at runtime, the value of variable `type` will specify the actual type of the subsequent embedded variable `inOutStruct`. We use the above-mentioned map and processed-data structure to implement this behavior.

Not all parts of a data structure need to be present in a message. Absent data structure elements are indicated by a nullpointer. The XML protocol variant serializes a nullpointer by simply omitting the tag for the corresponding element. Not packing any structure details, the Native variant of the protocol does not have this luxery. Instead it packs the UTF-8 representation of the string literal `"%@#ANULLSTR$%"` to depict a nullpointer. Note that, due to the iRODS protocol design, a serialized nullpointer cannot be distinguished from a serialized pointer to a string that bears a value equal to the nullpointer literal. In this odd case, we will interpret and unpack the structure as nullpointer.

Client API and data structures

iRODS communications follow a client/server model with typed messages. For instance, the first message sent by a client is a `RODS_CONNECT` type message, optionally followed by a `RODS_CS_NEG_T` message. Once connected, API calls use `RODS_API_REQ` type messages, and the server replies with `RODS_API_REPLY` type messages. When a client wishes to disconnect, it sends a `RODS_DISCONNECT` type message.

In general, the client sends a message, to which the server replies with a response message. We have encountered two exceptions to this rule. Firstly, the client must send two additional messages immediately after issuing a `RODS_CS_NEG_T` type request message, before it may expect a response. Secondly, when the client wants the session to end and sends a `RODS_DISCONNECT` type message, the server will not reply to this message. Instead, the client is expected to disconnect the socket. To support all use cases, including both exceptions, we make the client API layer responsible for message workflow decisions.

We design the client API as a central class `Irods`. Figure 5 demonstrates how Java applications instantiate this class and perform API calls. Each method of the `Irods` class represents an API call and returns the message component of a server reply as its result, transformed to Java classes more digestable for client applications. We customize the return type per method. For instance, the `rcMiscSvrInfo` call will return the received reply message as a `MiscSvrInfo` object. This approach facilitates compile-time type checking of client application calls. The remaining reply components, such as the errorMsg datastructure and the byteStream data, along with the `intInfo` returncode contained in the

```
Irods irods = new Irods(host, port);
RodsVersion version = irods.rcConnect(proxyUser, proxyZone, clientUser, clientZone);
if (irods.error) {
   throw new RuntimeException("Unable to connect, error is " + irods.intInfo);
}
MiscSvrInfo info = irods.rcMiscSvrInfo();
System.out.println("zone is " + info.rodsZone);
irods.rcDisconnect();
```

Figure 5. Example application using the client API

header, have a structure that is common across all API calls. They can be queried by client applications as object attributes.

Use case patterns

We have named our client library implementation "MyRods"[1]. While developing and testing MyRods, we encountered a recurring need for certain sequences of client-server interactions. For convenience reasons, we have captured these patterns in a few macro-level classes.

The class `Hirods` extends the API class `Irods`. It adds methods to unburden the client application of roundtrips involved with native or PAM authentication. In addition, authenticated sessions can be cloned, for instance to support parallel data transfers. These secondary connections are typically managed and reused via an `IrodsPool` class. The `Hirods` method `getAvus` allows applications to retrieve all object, resource or user related metadata, and likewise sets of metadata can be added to an entity using method `addAvus`. Further, the `checkAccess` method queries the server to check if a user has the desired access to a collection or data object, similar to, although more generic than, the iRODS microservice `msiCheckAccess` [10].

The class `DataTransfer` is an abstract class for copying the content of a file to another file. Using a strategy pattern, the source and destination files may be specified as either an iRODS based `Replica` or a regular filesystem file (`LocalFile`). Data transfers are not limited to *put* and *get* type operations between a file and a data object in an iRODS zone. They can also be used to copy data between data objects in the same zone or across zones. A second strategy pattern is deployed to influence the method used for data transfer, with implementations for single threaded transfer and multi-threaded transfer over the zone-port.

Recently, our institute has commenced to migrate and consolidate data located on several local iRODS zones to a single third-party managed iRODS instance. We have developed the application `ipump` [2], based on our MyRods client library and patterns, to support this migration process.

Performance

We benchmark the efficiency of the Native iRODS protocol variant against the XML variant in terms of exchanged message size.

Our test comprises of a variety of common calls: connect to an iRODS 4.3.3 server, perform a native login, request status information on a data object, query data object names and their data size from a collection that contains one hundred data objects, download the content of a data object, and disconnect. We run this test for both protocol variants in turn. In our initial connect message, we specify the iRODS protocol variant to be used for the remainder of the calls. Figure 6 outlines the header and message component size of iRODS messages that have been exchanged

[1]See https://github.com/tsmeele/MyRODS
[2]See https://github.com/tsmeele/ipump

Direction	iRODS Message type		Header (bytes)		Message (bytes)		Native
(client)	Type	Subtype	Native	XML	Native	XML	vs XML
sent	RODS_API_REQ	AUTH_REQUEST	132	132	0	0	100%
received	RODS_API_REPLY		140	141	64	153	69%
sent	RODS_API_REQ	AUTH_RESPONSE	133	134	29	119	64%
received	RODS_API_REPLY		139	139	0	0	100%
sent	RODS_API_REQ	OBJ_STAT	134	134	101	317	52%
received	RODS_API_REPLY		140	141	74	274	52%
sent	RODS_API_REQ	GEN_QUERY	133	134	81	423	38%
received	RODS_API_REPLY		142	142	2078	7787	**28%**
sent	RODS_API_REQ	GEN_QUERY	133	134	81	421	39%
received	RODS_API_REPLY		142	142	1116	3789	32%
send	RODS_API_REQ	GET_RESOURCE_INFO _FOR_OPERATION	136	136	112	328	53%
received	RODS_API_REPLY		140	141	76	150	74%
send	RODS_API_REQ	REPLICA_OPEN	136	136	104	321	53%
received	RODS_API_REPLY		142	142	1711	2345	75%
send	RODS_API_REQ	DATA_OBJ_READ	133	134	36	218	48%
received	RODS_API_REPLY		141	141	0	0	100%
send	RODS_API_REQ	DATA_OBJ_CLOSE	133	134	36	212	49%
received	RODS_API_REPLY		139	139	0	0	100%
sent	RODS_DISCONNECT		133	133	0	0	100%
		Total after connect	2601	2609	5699	16857	43%

Figure 6. Benchmark test results

after the initial connect.

The header component is always XML-serialized, regardless of the chosen protocol variant. This explains why there is almost no difference in size, except for one character in case the message component size, a value included in the header, requires an extra digit.

With regard to message component size, the Native protocol greatly outperforms the XML protocol, The difference is particularly striking in calls where many data values are being exchanged, such as a general query. Interestingly, while calls to transfer replica data do not exhibit any difference in size, the messages of the accompanying calls needed to open or close a replica display a factor two difference between the variants.

Overall, the Native protocol variant is able to pack messages by a factor two to nearly four more efficient compared to the XML variant, depending on the mix of API call types used.

DISCUSSION AND CONCLUSION

Our MyRods implementation of the client API library bears some limitations. The current set of implemented API calls only supports connections from a client application to an iRODS server, it does not support connections between servers. Parallel data transfers are only supported via the zone-port. We have not yet implemented so called *portal-based* transfer, which involves the use of high-ports, as we expect this method to be deprecated in a future iRODS release[3].

Our benchmark test demonstrates that the Native variant of the iRODS protocol packs messages two to four times

[3]see https://github.com/irods/irods/issues/7949

more efficient compared to the XML variant. As the Native variant is only available to C/C++ based applications, unfortunately it remains a hidden gem.

Through our design and implementation of a Java-based iRODS client API library, we have demonstrated that the Native variant can be supported in other languages besides C/C++. The implementation in Java consists entirely of Plain Old Java Objects (POJO) and should therefore be portable across many operating system architectures.

Initial design and prototype development of the library has taken approximately sixty hours of development time. We have built an initial prototype to confirm, and adjust, our interpretation of the iRODS protocol specification. Once the lower two layers were established, we continued for another several weeks to complete the client API layer and support for the use case patterns.

Our approach, to internally use data structures that mimic the iRODS types, allowed for a rapid and consistent implementation of message serialization. Modeling of pointers as a separate data type turned out to be an important insight that simplified the serialization of elements.

Adding support for API calls required extensive inspection of iRODS server source code. The iRODS API is mostly documented inline in the source code. Sometimes information is fragmented as a result of a change in development practices since the introduction of the iRODS plug-in architecture. We recommend that iRODS returns to a more centralized definition of pack instructions and keywords, so that they are easier to locate and reuse. In addition, more complete, findable documentation on semantics of iRODS API arguments would be helpful for client application developers.

We learned that replicas share sufficient properties with regular files, to be modeled as specializations of a common supertype. More research will be needed to investigate the implications for high level client interfaces to iRODS. Can we simplify existing interfaces by using a different paradigm? For instance, should we continue to think of the entire iRODS zone as a filesystem? Or would it instead be appropriate to consider (and expose) each of its resources as separate filesystems?

REFERENCES

[1] M. Wan, R. Moore, and A. Rajesekar, "Distributed Shared Collection Communication Protocol." https://www.irods.org/index.php/iRODS_Protocol_Overview, May 2008.

[2] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The SDSC storage resource broker," in *CASCON First Decade High Impact Papers*, pp. 189–200, IBM Press, 2010.

[3] M. Stonebraker and L. A. Rowe, "The design of Postgres," in *Proceedings of the 1986 ACM SIGMOD international conference on Management of data*, vol. 15, pp. 340–355, ACM New York, NY, USA, 1986.

[4] M. Conway, "Enhancing iRODS Integration: Jargon and an Evolving iRODS Service Model," in *Proceedings iRODS User Group Meeting 2010*, (Chapel Hill, NC, USA), pp. 62–66, DICE, 2010.

[5] DICE-UNC, "PHP API for iRODS." https://github.com/DICE-UNC/irods-php.

[6] iRODS Consortium, "Python iRODS Client." https://github.com/irods/python-irodsclient.

[7] I. Choi, J. H. Hartman, and E. Skidmore, "Go-iRODSClient, iRODS FUSE Lite, and iRODS CSI Driver: Accessing iRODS in Kubernetes," in *iRODS User Group Meeting 2021 Proceedings*, (Virtual), pp. 63–72, iRODS Consortium, 2021.

[8] J. Reynolds and J. Postel, "Assigned numbers," RFC 1700, RFC Editor, Oct. 1994.

[9] J. Klensin and M. Padlipsky, "Unicode Format for Network Interchange," RFC 5198, RFC editor, Mar. 2008.

[10] A. Rajasekar, T. Russell, J. Coposky, A. de Torcy, H. Xu, M. Wan, R. W. Moore, W. Schroeder, S.-Y. Chen, M. Conway, and J. H. Ward, "iRODS 4.1 Microservices Workbook." https://irods.org/uploads/2015/01/irods4-microservices-book-web.pdf, May 2015.

iRODS HTTP API v0.5.0

Martin Flores, Kory Draughn, Terrell Russell
iRODS Consortium

ABSTRACT

The iRODS HTTP API has had two releases this past year. Updates include more iRODS API coverage, better logging, and an OpenID Connect plugin framework.

iRODS UGM 2025, June 17-20, 2025, Durham, NC

AI Verde MCP Server: Bridging Generative AI with CyVerse Data Resources

Illyoung Choi, Edwin Skidmore, Nirav Merchant
CyVerse / University of Arizona

ABSTRACT

This talk introduces the AI Verde MCP Server, a new Model-Context-Protocol (MCP) Server being developed to extend the capabilities of AI-VERDE—an integration platform designed to support research and educational teams working with generative AI technologies. AI-VERDE provides secure and flexible access to a wide range of commercial and open-source LLMs, including GPT-4, Claude, and LLaMA4, as well as inference service providers supported by the NSF, such as Jetstream2.

The AI Verde MCP Server enables integration with data and computing services within the CyVerse infrastructure. Its initial focus is on providing access to the CyVerse Data Store, a research data repository built on the iRODS data management system. Operated for over a decade, the Data Store hosts a wide variety of research data across scientific domains. By connecting this data to AI systems through the MCP Server, researchers will be able to explore new methods of analyzing stored datasets using large language models. Future plans include extending this integration to the CyVerse Discovery Environment.

This talk will share the architecture and development progress of the AI Verde MCP Server, and explore how connecting AI-VERDE to CyVerse resources can enable practical, data-driven AI applications in research and education.

iRODS UGM 2025, June 17-20, 2025, Durham, NC