iRODS®

Amsterdam

USER GROUP MEETING
2024 PROCEEDINGS

PUBLISHED BY THE iRODS CONSORTIUM

iRODS
User Group Meeting 2024
Proceedings

© 2024 All rights reserved. Each article remains the property of the authors.

16TH ANNUAL CONFERENCE SUMMARY

The iRODS User Group Meeting of 2024 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 May 28th to 31st, hosted by SURF in Amsterdam, Netherlands with 131 people attending from 13 countries. Attendees and presenters represented over 49 academic, governmental, and commercial institutions.

TALKS AND PAPERS

LIGHTNING TALKS

Helping users keep filesystems clean

Emyr James – Centre for Genomic Regulation (CRG)

iRODS Dataverse Integration

Danai Kafetzaki – KU Leuven

Bridging iRODS and supercomputing for RDM-driven data-to-compute

Mher Kazandjian – SURF

iRODS: A Financial Perspective

Jan de Graaf – Netherlands Cancer Institute (NKI)

iRODS Testing K8s Demo

Phil Owen – Renaissance Computing Institute (RENCI)

Metadata and Data Landscape Report - Demo

David Cerf – GRAU DATA

iBridges Update and Demo

Raoul Schram – Utrecht University

iRODS replication management and our future iRODS world at Wellcome Sanger Institute

Sai Poomdla
Wellcome Sanger Institute

ABSTRACT

At Wellcome Sanger Institute, we manage 20 Petabytes of genomic data using iRODS. We currently use iRODS 4.2.7 in our production (and our tales of upgrading it to a later version) and due to various scenarios, whether it might be in hiccup in the network connection or data object corruption in the transit, we end up with failure in our replica creation process leading to Single, Dirty and Triple replicas. To mitigate these issues, we have developed in-house solutions using python iRODS client. Also, we are evolving into a new iRODS world, where we want to replace our storage servers with single large file-system and allow our users directly to read from this file-system instead of downloading the files into a high-performance file system.

iRODS UGM 2024, May 28-31, 2024, Amsterdam, Netherlands
Author(s) retain copyright.

Python package for metadata schemas

Mariana Montes, Ronny Moreas
KU Leuven

ABSTRACT

In this talk we will present a Python package to validate, write and read structured metadata based on schemas. The schemas are described in a JSON file following the specifications of the metadata schema manager developed in the context of the ManGO portal, but this implementation is independent from the web application. Given a schema, users can provide hierarchical metadata in a Python dictionary, validate its contents and add it to an iRODS data object or collection as AVUs with the appropriate namespacing. The same package allows to read back the AVUs and parse them into a Python dictionary preserving the original hierarchy.

iRODS UGM 2024, May 28-31, 2024, Amsterdam, Netherlands
Author(s) retain copyright.

iRODS-based system turbocharged next-gen sequencing analysis during pandemic and beyond

Robert Verhagen, Erwin van Wieringen
Dutch National Institute for Public Health
and the Environment (RIVM)

ABSTRACT

The National Institute for Public Health and the Environment (RIVM) has numerous projects in various scientific domains that generate next generation sequencing data. Bioinformatics plays an important role in analysing and interpreting this sequencing data. To support these analyses we developed a platform that consists of a High Performance Compute (HPC) cluster, a Linux Scientific Workspace for software development and a Data Management System (DMS) based on iRODS. On top of this DMS, we also created a Job Engine: a tightly integrated process automation tool that manages the automated analyses of sequencing data on the HPC.

The development of this robust and scalable platform started two years prior to the COVID-19 pandemic. As the pandemic unfolded, the RIVM found itself tasked with extensive analyses of vast amounts of COVID-19 surveillance data. We were able to quickly and seamlessly scale the system up to meet this increased demand.

For better automation and user experience, we created additional components within and alongside iRODS. The aforementioned Job Engine and a tool to import data from a variety of internal and external sources are used for automation purposes, while an extensive web interface with collection and document viewer functionalities, a data lineage viewer and (meta)data search capabilities were added for easy user interaction. We also created rules to pack and archive iRODS collections to an iRODS consumer at SURF (collaborative organisation for IT in Dutch education and research). With the DMS and these additional components, we have made the data analysis component of many vital processes within the RIVM reproducible.

By leveraging the iRODS software both the efficiency and the quality of data analyses within the RIVM has been improved significantly. The platform has proven to be crucial, notably during the pandemic, for processing otherwise unimaginable amounts of data.

iRODS UGM 2024, May 28-31, 2024, Amsterdam, Netherlands
Author(s) retain copyright.

iRODS Build and Test v9: Automation via GitHub and Kubernetes

Phil Owen
Renaissance Computing Institute (RENCI)

Terrell Russell, Kory Draughn, Alan King
iRODS Consortium

ABSTRACT

A reliable and efficient testing environment is vital to foster the quality of iRODS products and enhance developer productivity. It has become a moderately complex and time-consuming undertaking to create, maintain, and execute testing suites across numerous iRODS environment variants in order to comprehensively test iRODS source code.

Typically, unit and topology testing of iRODS components begins with building packages from modified source code. Next, iRODS packages are installed on various target environments that are defined within a matrix of versioned operating systems and DBMS types. Finally, test suites are executed in each environment where log files of all test results are compiled and analyzed.

In this talk we will discuss our newest approach (and future vision) of automating iRODS testing using GitHub and Kubernetes. This discussion will cover some details on how developers will request testing runs, how the build/test process works, and the forensics performed on the test results.

iRODS UGM 2024, May 28-31, 2024, Amsterdam, Netherlands
Author(s) retain copyright.

Sharing data in a multi-system multi-role environment centered on iRODS

Claudio Cacciari
SURF

Eduard Klapwijk, Simone Mulder, Mark Mulder, Stephan Heunis, Frans van der Zijde, Wouter Timmer
Erasmus University Rotterdam

ABSTRACT

SURF, the cooperative association of Dutch educational and research institutions, offers data infrastructure and services to the research communities. Some of its services are based on iRODS and are often used as building blocks for data platforms. One increasingly common architectural component in those platforms is a web portal where researchers can discover data using project specific queries. Once the data are found, they are made available to the researcher, directly, for example, with a download link or indirectly, triggering a copy to a computing environment where they are analyzed. The implementation of such workflow is time consuming. Its maintenance in the long term is often jeopardized by limited support available within the project and design choices too tailored for that use case makes its adoption by other organizations too difficult. We think that it is possible to model that workflow in a generic way as a re-usable modular component and in a way flexible enough to support even the more stringent requirements associated with sensitive data. The component relies on iRODS and links together multiple web portals and repositories through an API layer based on FastAPI. We present here a proof of concept developed within the GUTS project, in collaboration with the project's data management team and the research support.

iRODS UGM 2024, May 28-31, 2024, Amsterdam, Netherlands
Author(s) retain copyright.

iRODS Security Challenges Within an Enterprise Environment

Ryan Blome, Simon Cook
Dow

ABSTRACT

Dow's focus on data security necessitates a tailored approach for our internal users, leading to the development of the Scientific Data Management System (SDMS) Query Tool (SQT) — a user-friendly tool designed to facilitate secure access to specific datasets. The current gap with Metalnx for general users is that there is too much control over modifying data and collections. Additionally, it is difficult to synchronize the iRODS users to our existing Azure Security groups for permission management. This talk outlines the development of a Querying Tool utilizing the iRODS C++ API as a backend to communicate with iRODS. The talk will highlight the need for robust security architecture for Enterprise scale applications and where we are hoping to take the project to in the future (HTTP API, etc.)

iRODS UGM 2024, May 28-31, 2024, Amsterdam, Netherlands
Author(s) retain copyright.

Testing iRODS-based applications: Experiences with Yoda

Claire Saliers
Utrecht University
c.l.saliers@uu.nl

Lazlo Westerhof
Utrecht University
l.r.westerhof@uu.nl

Sietse Snel
Utrecht University
s.t.snel@uu.nl

Sirjan Kaur
Utrecht University
s.kaur@uu.nl

ABSTRACT

Yoda is a research data management system based on iRODS that enables researchers to deposit, share, publish and preserve data. It is developed at Utrecht University, and used at multiple universities, as well as other organizations. In this contribution, we share our approach for testing Yoda and planned future work.

Our testing approach primarily revolves around automated scripted testing, encompassing unit tests, integration tests, API tests and UI tests. Additionally, we have explored use of scriptless testing and fuzzing to supplement scripted tests.

Planned future work includes improving test coverage, introducing accessibility tests and exploring ways to improve reliability of UI tests.

Drawing from our experiences, we share recommendations regarding how to improve testability and reliability of iRODS-based applications.

Keywords

Yoda, software testing, software quality assurance, iRODS

INTRODUCTION

In 2014, Utrecht University commenced development of Yoda [9], an open-source research data management system built upon iRODS [8] as its foundational component. Yoda customizes iRODS with specific domain logic and adds a user-friendly graphical interface. It supports multiple workflows and metadata schemas to address the requirements of various scientific disciplines. Since its inception, Yoda has undergone continuous enhancement, introducing a range of features to empower researchers in describing, depositing, and publishing their research data in alignment with the FAIR principles [15].

Within Yoda lie the crown jewels of academic research, preserved and accessible for reuse. This paper delves into the role of *automated* testing in preserving and managing these assets in a sustainable and maintainable manner.

Earlier publications regarding software testing in combination with iRODS have generally focused on testing iRODS itself, e.g. [4]. In this paper, we focus more on testing applications built on iRODS, based on our experiences with testing Yoda.

The rest of this paper is structured as follows. The next section has background information about software testing in general. After that, we discuss our approach for automated tests of Yoda, and we report preliminary results of our experiments with unscripted testing. Finally, we share lessons learned in the process of improving the way we test Yoda.

iRODS UGM 2024 May 28-31, 2024, Amsterdam, Netherlands
[Copyright Utrecht University, 2024]

SOFTWARE TESTING

Software testing is an essential part of the software development life cycle that aims to analyse and improve quality, performance, security, and other application attributes. Testing is a crucial process to ensure that the application functions as expected. Software testing helps to achieve the following objectives [3]:

1. **Risk Mitigation** - Testing can assist in mitigating risks by identifying undesirable behavior of the application early in the development process.

2. **Quality and Compliance** - Testing can ensure that the application adheres to regulations, standards, and compliance requirements set by the industry and organizations.

3. **User satisfaction** - A positive user experience is of utmost significance for the success of an application. Testing from a user's perspective can give an insight into the usability aspect of the application. Developers can then make corresponding improvements to improve this aspect.

4. **Optimization** - Testing can provide vital feedback to continuously improve the functionality, quality, performance, and security of an application.

Pareto principle

For ever-evolving and complex applications, testing must be time-efficient. The Pareto principle helps to have a more focused approach in software testing.

The Pareto principle states that 80 % of the results stem from 20% of the effort. The 80:20 pattern was discovered by economist Vilfredo Pareto in the 1880s. He concluded that the wealth of the Italian population followed an 80:20 rule (80% wealth for 20% of people) and suggested that this rule could be found in many other areas.

In today's world, software applications are constantly evolving and expanding to suit the needs of users. It is increasingly expensive to invest in software testing. The Pareto principle in software testing suggests that 80% of the defects or failures originate from 20% of the code. The numbers are approximate. It could also be a 70:30 ratio [17]. Nevertheless, this rule helps concentrate on specific areas of the application for rigorous testing. For example, one might concentrate on first testing the sections of code that are used the most frequently, or a section of code that controls a feature that is highly valued.

Testing Strategies

Software testing strategies differ per organization. These strategies include methods of testing different levels of software application. The testing methodologies can be broadly divided into [13]:

1. **Unit Testing** - Unit testing is testing the smallest unit of code. It is performed early in the development stage to test the functionality of each individual component.

2. **Integration Testing** - Once all components of an application are unit tested, they are integrated as a group and then tested. Additionally, integration testing involves testing the interface between two or more software units or modules.

3. **E2E Testing** - End-to-end testing involves simulating real-world user experiences with the application. These tests are performed when all the components pass the integration tests.

The main testing strategy for Yoda is described in the following section.

TESTING APPROACH FOR YODA

The main strategy for testing Yoda is to try to test the main parts of the code, testing at several layers with mainly automated tests. We aim to cover at least the expected situations, and also common errors, but not with a specific coverage percentage in mind. We use Pytest [5] and unittest [14] as test frameworks, Splinter [11] for web tests, and Coverage.Py [2] for test coverage measurement. Our unit, API, UI, integration, and smoke tests are scripted, automated tests.

At the lowest level, we use unit tests to check the basic functionality of individual pieces of code. Unit tests do not have any iRODS dependencies and thus can be run in many different environments easily.

For the API and UI tests, we have a test playbook that sets up the situations for the tests. It creates the users and groups needed for the tests. Both API tests and UI tests are grouped into topics, typically around certain modules or functionalities of Yoda. We try to make these groups of tests self-contained, so that if all tests in the group are run together, they will not affect future tests, and can be run repeatedly if run together. For example, in the group manager tests one group is created, and then one of the last tests is to delete that group. This also allows us to only test one part of Yoda if our changes only affect that one part. This principle of self-containment is not always practical to follow so we do not follow it strictly.

For the ruleset portion of our code, we write our rules in Python partly to make testing easier. We run integration tests to test that our Python code works with iRODS, using a Python rule that runs each test. For example, we have a Python function to check that a data object exists, and we run an integration test to test it.

How and when tests are run

Unit, integration, API, smoke, and UI tests can be run locally. We run these tests locally and frequently during development. Integration, API, smoke, and UI tests are held centrally in one repository, with options to enable and disable certain categories of tests when run.

Unit, integration, and API tests are run for every pull request, and in some cases for every commit made to our GitHub repositories. These are run automatically using GitHub Actions. This allows us to confirm functionality before adding new code to our development branch. The UI tests do not always reliably pass and are only run locally.

Smoke tests are a small set of tests to confirm that a Yoda environment has been deployed or updated correctly, while creating very few groups and users. This is useful to run on acceptance environments, where we want to change the basic setup as little as possible but still confirm that the environment is configured correctly.

Manual testing

Manual acceptance tests are run before a new release of our software. We perform manual acceptance testing for a few different reasons. One is to ensure that the code works well in combination with our acceptance and production infrastructure, since much of our automated tests run in development environments. Another is to test components that do not have good automated test coverage yet, either because it is a difficult situation to test or we have not had time to create a test for it. Some aspects are difficult to test by automated tests, such as visual and general layout issues.

There is a list of required functionality to test manually, many overlap with our API and UI tests. For example, there is a test to create a new research group with a new category and subcategory. This can help us catch any unexpected changes in the UI during this action that our automated UI tests might not have registered. There is another test to browse a folder and change the number of folders shown; in addition to checking for visual issues we can check that the folder loads in an acceptable amount of time for the user.

FUZZING AND SCRIPTLESS TESTING

The regular test suites of Yoda focus on detecting problems by examining the response of the system to pre-determined inputs. That is, we mainly test how the application responds to inputs that we expect.

In practice, we have noticed that bugs also commonly relate to unexpected input values. For example, the application can respond in an unexpected way if entity names (e.g. collection names, user names) contain unexpected characters, or if they hit an unexpected edge case related to their length. Although we can use scripted tests to test for regressions after we have fixed such a bug, detecting such issues proactively using scripted tests is difficult in practice.

We have therefore also experimented with supplementing our scripted tests with two non-scripted approaches:

1. We have tried to assess robustness of iRODS rules by invoking them repeatedly with randomized input values (*fuzzing*)
2. We have tried to test robustness at the GUI level by using scriptless GUI testing, which involves traversing the GUI of an application in an unscripted way, and detecting unexpected error conditions.

Fuzzing iRODS rules

Fuzzing has been defined as "the execution of the [program under test] using input(s) sampled from an input space [...] that protrudes the expected input space of the [program under test]" [6]. That is, fuzzing involves testing a program with inputs that are possibly unexpected. Inputs that cause undesirable behaviour, such as crashing the program, are reported. Fuzzing is commonly used in security testing, as well as testing software for robustness.

In the context of iRODS, examples of undesirable behaviour of software components related to unexpected inputs could be:

- Rules or microservices should not crash the iRODS agent process if they receive unexpected inputs, but rather return an error code or error status.
- Rules or microservices should log a descriptive concise error message in case of unexpected inputs, rather than a series of low-level error messages (e.g. *function_foo: collection name must be an absolute path*, rather than a stack trace or SQL error). This makes diagnosing problems on production systems, which generally have high rates of log messages, easier.

The iRODS rule fuzzer is an experimental Python-based tool that applies fuzzing to iRODS rules [10]. An overview of the fuzzing process is presented in figure 1.

The fuzzer needs information regarding which endpoints (rules or microservices) to test. The current version implements two endpoint discovery methods. Firstly, a user can provide a list of endpoints to be tested using a CSV file. This method should be usable for any iRODS-based application. Secondly, the fuzzer can try to discover rules directly in the ruleset. The current implementation of this method is specific to Yoda. The fuzzer parses Python rule files using the `ast` module. The fuzzer then retrieves names of functions that have an `api.make` or `rule.make` decorator, which are the most common ways to define a rule in Yoda [16]. For legacy rule language files, the fuzzer uses a regular expression to find rules.

Once a list of endpoints has been compiled, the fuzzer may need to gather additional information regarding the endpoint parameters. For example, in some cases the fuzzer may need to determine which parameters are input parameters and which ones are output parameters. The fuzzer currently accomplishes this by simply trying all possible combinations of input and output parameters, and monitoring error codes to determine which combination is accepted.

Now the fuzzer has a list of endpoints, as well as information about their parameters. The fuzzer will now enter a loop. The first step is to generate input argument values. The present version of the fuzzer uses randomly generated values. The second step is that the input argument values are transformed, if needed. For example, Yoda API rules need to receive their input values in JSON format. Thirdly, the rule is executed with the input argument values and the result is logged.

After running the fuzzer for some time, the user can examine the rodsLog to see which inputs resulted in agent crashes or other unexpected error conditions. We have used fuzzing results to identify areas of improvement in our rulebase, such as rules that needed to have better input validation.

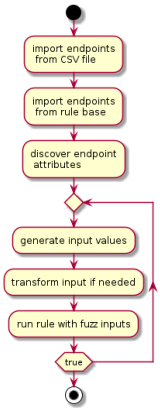

Figure 1. Process diagram of the experimental iRODS rule fuzzer.

Scriptless GUI testing

In additional to testing rules for undesirable behaviour when they receive unexpected inputs, we have also explored testing the GUI by providing it with unexpected inputs. To this end, we use Testar [12], a software tool for scriptless GUI testing.

The basic process of Testar is similar to fuzzing: it continuously provides a system under test (such as a web application) with GUI inputs and monitors the system for unwanted behaviour. The baseline process is to randomly select GUI actions (such as clicking on a button, or entering text in a text field) and execute them. This behaviour can be customized in custom settings (a *protocol*) for a specific application. For example, the protocol could contain code to log in before starting tests, and to avoid clicking on the log out button. The process can also be customized in other ways, which are outside the scope of this paper.

An example of undesirable behaviour that can be detected using Testar is an *internal error* message in the GUI. Internal errors should not occur in response to unexpected actions by a user. In such cases, the application needs to either prevent the user from making a mistake (e.g. by not letting the user submit a form before they have filled in compulsory fields) or give a more descriptive message (e.g. "you need to enter at least one author when submitting a data package").

BEST PRACTICES AND LESSONS LEARNED

We share a list of our lessons learned in the process of improving Yoda below:

1. All else being equal, preferably write rules in Python rather than in the legacy rule language. Using a language that has standard tools for unit testing and mocking objects makes it much easier to test code automatically. The availability of other code quality tools, such as linters, is also helpful.

2. When it is necessary to prioritize what tests are written when developing a new feature, consider prioritizing unit tests and low-level integration tests over API and UI tests. It is better to add unit tests and low-level integration tests sooner rather than later. Adding these tests to existing code often requires refactoring, so it may take more time to add them later. By contrast, adding API and UI tests to functionality that has already been developed usually does not require extra work, and can more easily be added as needed rather than immediately on feature creation.

3. Mixing domain logic with code that performs iRODS operations often results in code that is difficult to test automatically. That is, the code that interacts with iRODS gets in the way of writing simple unit tests for the domain logic, and the domain logic gets in the way of writing simple integration tests for the code that interacts with iRODS. This can be avoided using dependency inversion, or by extracting the domain logic into separate functions.

   Consider the code samples below, which are simplified samples of the code that Yoda uses to determine whether it needs to store a revision of a data object in the revision store. The first fragment shows the structure of the old code, where iRODS operations and domain logic are mixed, whereas they are separate in the second fragment. The second version makes it easier to add unit tests for the logic in the *is_eligible_for_revision* function, since this function only depends on a few small utility functions.

Listing 1. Old code with mixed domain logic and iRODS operations (edited)

```
def resource_modified_post_revision(ctx, resource, zone, path):
    size = get_dataobject_size(path)

    # A revision should not be created when the data object is too big
    if int(size) > get_max_size_for_revision():
        return

    # Only create revisions for research space
    if not path.startswith(f"/{zone}/home/research-")
        return

    if basename(path) in get_object_names_exempt_from_revision():
        return

    msi.add_avu(ctx, '-d', path, "need_to_create_revision", resource)
```

Listing 2. New code with extracted domain logic (edited)

```
def resource_modified_post_revision(ctx, resource, zone, path):
    size = get_data_object_size(path)
    if is_eligible_for_revision(size, path, zone):
        schedule_for_revision(ctx, path, resource)

def schedule_for_revision(ctx, path, resource):
    msi.add_avu(ctx, '-d', path, "need_to_create_revision", resource)

def is_eligible_for_revision(size, path, zone):
    if size > get_max_size_for_revision():
        return False
    elif not path.startswith(f"/{zone}/home/research-"):
        return False
    elif basename(path) in get_object_names_exempt_from_revision():
        return False
    else:
        return True
```

FUTURE WORK

We plan to explore improving our test setup in the following ways:

- Test coverage in Yoda is currently still uneven across components and functionality. Although some parts have satisfactory coverage, we still have to work on parts of the application that have limited or no automated test coverage.
- The UI tests are flaky on some systems. We can therefore not yet run them automatically in CI. We plan to experiment with using Playwright [7], which claims to enable more reliable testing of web applications.
- Our current testing approach does not include accessibility testing [1]. We plan to explore how we can incorporate this into our testing strategy, preferably in an automated way.

REFERENCES

[1] Ara, J., Sik-Lanyi, C. & Kelemen, A., *"Accessibility engineering in web evaluation process: a systematic literature review"*, Univ Access Inf Soc, `https://doi.org/10.1007/s10209-023-00967-2`, 2023.

[2] Batchelder, Ned, and Contributors to Coverage.py, *Coverage.py: The code coverage tool for Python*, Repository: `https://github.com/nedbat/coveragepy`, Documentation: `https://coverage.readthedocs.io`.

[3] Bourque, P., and Fairley, R.E., eds., *Guide to the Software Engineering Body of Knowledge*, Version 3.0, IEEE Computer Society, `www.swebok.org`, 2014.

[4] King, Alan and Russell, Terrell, *iRODS Development and Testing Environments (v8)*, iRODS User Group Meeting 2022 Proceedings, p75–80, see `https://irods.org/uploads/2022/irods_ugm2022_proceedings.pdf`, 2022.

[5] Krekel et al., *PyTest*, `https://github.com/pytest-dev/pytest`

[6] Manès, V. J. M., et al., *The Art, Science, and Engineering of Fuzzing: A Survey.* In: IEEE Transactions on Software Engineering, vol. 47, no. 11, pp. 2312–2331, `https://doi.org/10.1109/TSE.2019.2946563`, 2021.

[7] *"Playwright"*, `https://playwright.dev/`.

[8] Rajasekar, Arcot and Moore, Reagan and Hou, Chien-yi and Lee, Christopher A and Marciano, Richard and de Torcy, Antoine and Wan, Michael and Schroeder, Wayne and Chen, Sheau-Yen and Gilbert, Lucas and others, *"iRODS primer: integrated rule-oriented data system"*, Synthesis Lectures on Information Concepts, Retrieval, and Services, vol. 2, no. 1, pp. 1–143, 2010.

[9] Smeele, Ton and Westerhof, Lazlo and Smeele, Chris and van Schip, Rob and Croes, Felix A. and Hoogerwerf, Maarten and Kleinloog, Hans and Alebregtse, Jurgen and van de Hoef, Rick and de Mooij, Jan and de Raaff, Harm and van Elk, Roy and Frederiks, Paul and de Ruiter, Joris and Snel, Sietse and Zondergeld, Jelmer and Kaur, Sirjan and Saliers, Claire, *"Yoda: research data management"*, `https://github.com/utrechtuniversity/yoda`, 2024.

[10] Snel, Sietse, *irods-rule-fuzzer: an experimental fuzzer for iRODS rules.* `https://github.com/utrechtuniversity/irods-rule-fuzzer`

[11] Splinter `https://github.com/cobrateam/splinter`

[12] Vos, T.E., Aho, P., Pastor Ricos, F., Rodriguez-Valdes, O. and Mulders, A., *Testar – Scriptless Testing Through Graphical User Interface.* In: Software Testing, Verification and Reliability, 31(3), p.e1771, `https://doi.org/10.1002/stvr.1771`, 2021.

[13] Umar, M., *Comprehensive study of software testing: Categories, levels, techniques, and types*, Vol. 5, Issue 6, International Journal of Advance Research, Ideas and Innovations in Technology, `https://www.ijariit.com/manuscripts/v5i6/V5I6-1154.pdf`, 2019.

[14] *unittest — Unit testing framework*, In: Python documentation, `https://docs.python.org/3/library/unittest.html`

[15] Wilkinson, Mark D and Dumontier, Michel and Aalbersberg, IJsbrand Jan and Appleton, Gabrielle and Axton, Myles and Baak, Arie and Blomberg, Niklas and Boiten, Jan-Willem and da Silva Santos, Luiz Bonino and

Bourne, Philip E and others, *"The FAIR Guiding Principles for scientific data management and stewardship"*, Scientific data, vol. 3, https://doi.org/10.1038/sdata.2016.18, 2016.

[16] Yoda team, *Yoda and the iRODS Python Rule Engine Plugin.* In: Yoda technical documentation. See https://utrechtuniversity.github.io/yoda/design/other/python-plugin.html

[17] Yogi, M., Kumar, G., and Uma, D., *Iterative Pareto Principle for Software Test Case Prioritization*, Vol. 5, Issue 8, International Journal of Advanced Research in Computer and Communication Engineering, https://www.ijarcce.com/upload/2016/august-16/IJARCCE%2050.pdf, 2016.

[18] Zhao, X., Qu, H., Xu, J., Li, X., Lv, W. Wang, G., *"A systematic review of fuzzing"* Application of Soft Computing 28:5493–5522, https://doi.org/10.1007/s00500-023-09306-2, 2024.

iRODS Build and Packaging: 2024 Update

Markus Kitsinger
iRODS Consortium

ABSTRACT

This talk will provide an update on our journey to 'Normal and Boring' with regard to CMake, libstdc++, and building for various platforms.

iRODS UGM 2024, May 28-31, 2024, Amsterdam, Netherlands
Author(s) retain copyright.

iRODS HTTP API v0.3.0 with OpenID Connect

Kory Draughn, Martin Flores
iRODS Consortium

ABSTRACT

The new iRODS HTTP API was introduced last year as an idea to increase developer accessibility to an iRODS namespace, along with early research into how OpenID Connect would fit within the iRODS ecosystem. This talk will share updates through the first three releases, including optimizations and having the iRODS server be an OpenID Connect Protected Resource.

iRODS UGM 2024, May 28-31, 2024, Amsterdam, Netherlands
Author(s) retain copyright.

Safeguard your sensitive data in iRODS using data encryption feature available in GoCommands

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

ABSTRACT

iRODS has been utilized across various scientific domains for storing and sharing research data. However, certain domains, such as health sciences, require stringent confidentiality measures for utilizing iRODS as their data management solution. Compliance with regulations such as HIPAA (Health Insurance Portability and Accountability Act) necessitates end-to-end encryption, meaning that data must be encrypted for both storage and transmission. While iRODS provides encryption in transit using SSL (Secure Socket Layer), the responsibility for encrypted data storage at rest lies with the iRODS infrastructure provider and presents a significant limitation. Encrypting data before uploading it and decrypting it after download adds additional steps and complexity for manual encryption and decryption process.

To address these multiple challenges and to meet the requirement for encrypted data storage, we have simplified the process of encrypting and decrypting data as built-in capability in GoCommands. GoCommands is a cross platform command-line utility for iRODS, offering data access and management capabilities similar to iCommands. The encryption at rest algorithm employs AES-256-CTR to encrypt both file names and content before uploading, thereby ensuring that data remains encrypted within iRODS. When listing and downloading encrypted files, they are automatically decrypted using the encryption key provided by the user. Since GoCommands and WinSCP share the same encryption algorithm, users who are not familiar with the command-line interface can utilize WinSCP, a GUI-based tool, making it more user-friendly. This end-to-end encryption functionality requires no server-side changes to iRODS, ensuring compatibility with existing iRODS servers.

This new data encryption feature will empower researchers to utilize iRODS for storing and accessing their sensitive data and meeting regulatory compliance, thereby facilitating broader adoption of iRODS across various research domains.

iRODS UGM 2024, May 28-31, 2024, Amsterdam, Netherlands
Author(s) retain copyright.

ManGO Portal (iRODS PRC based web UI): Feature updates, including extension points and workflows

Paul Borgermans, Mariana Montes, Ingrid Barcena Roig
KU Leuven

ABSTRACT

As our Research Data Management user base is growing, so is our in-house software stack including ManGO, the iRODS PRC based web portal. Over the past year, more extension points were added in order to tailor the functions and user experience along the various requirements sets. Also a more generalised base to build workflows / data processing pipelines was added that can be triggered by events inside iRODS as well as external drivers (including ad hoc workflows).

Besides these rather fundamental aspects that make ManGO usuable for many iRODS installations, many smaller features and improvements were added as well.

iRODS UGM 2024, May 28-31, 2024, Amsterdam, Netherlands
Author(s) retain copyright.

Optimizing Data Management for AI: The Synergistic Role of Metadata-Hub and iRODS

David Cerf
GRAU DATA GmbH
david@graudata.us

ABSTRACT

Metadata-Hub, when combined with iRODS, becomes a transformative solution for managing escalating volumes of machine-generated and unstructured data. It places embedded metadata at the heart of storage management and data preparation, offering a more efficient, cost-effective, and scalable approach. At the time of data creation, Metadata-Hub extracts embedded metadata, allowing raw data files to be immediately archived to affordable storage solutions. This drastically cuts the need for expensive primary storage. Metadata-Hub becomes the repository for all metadata and makes it readily available. It reduces data preparation time for computational applications, analytics, and artificial intelligence (AI), accelerating the transition from data acquisition to insight derivation. Furthermore, the integration of Metadata-Hub into iRODS provides a robust, contextually-aware framework for automated and intelligent data workflows driven by embedded metadata, including archiving, replication, and distribution.

Keywords

Embedded Metadata, Data Warehouse, Data Management, Data Archive, Reduce Storage Costs, Data Preparation, Machine-Generated Data, Data Analytics, AI, Data Lakes, Data Fabric, Decentralized Storage

INTRODUCTION

In an era marked by an unparalleled increase in data generation, especially from machine-generated sources, organizations are confronted with escalating challenges: soaring storage costs and intricate data preparation processes. Metadata-Hub emerges as a revolutionary solution, emphasizing the critical role of embedded metadata in transforming data storage and preparation practices. This strategic innovation empowers organizations to harness their data assets with unmatched efficiency and effectiveness.

TRANSFORMING MACHINE-GENERATED DATA MANAGEMENT

A significant portion of the unstructured data within organizations is attributed to machine-generated outputs from various scientific instruments. This type of data is imbued with embedded metadata, a treasure trove of insights and essential information pivotal for sophisticated computational analysis, AI applications, and analytics. However, conventional storage methods, which entail retaining massive volumes of data on premium, high-cost disk tiers, not only incur substantial expenses but also obstruct prompt access to and analysis of data. This predicament arises from prevailing workflows that require maintaining data on expensive storage to accommodate data preparation needs. Consequently, individuals often engage in laborious manual efforts to sift through files to extract data critical for computational tasks. Moreover, this procedure often necessitates the transfer of vast data volumes across networks, exacerbating the delay in data preparation and escalating costs due to increased network traffic and time investments. Reliance on manual intervention for sorting and accessing files on costly storage solutions significantly detracts from operational efficiency, diminishes data quality, and postpones analytical endeavors.

The Metadata-Hub Innovation

Addressing these challenges head-on, Metadata-Hub revolutionizes data management by extracting embedded metadata at the inception of data creation or ingestion of the file to the storage. This groundbreaking approach ensures the capture and storage of all vital analytics and AI information independent of the raw data files, facilitating their immediate archival to more economical storage options like cloud archives or tape systems. This strategic architectural design drastically cuts the necessity for high-priced primary storage and significantly reduces the complexity associated with large-scale data management.

[Author retains copyright. GRAU DATA GmbH.]

HOW IT WORKS

Metadata-Hub elegantly converts unstructured data into a structured, searchable format, profoundly enriching its analytical and decision-making utility. Through direct integration with storage systems across SMB, NFS, and S3 protocols, Metadata-Hub's unparalleled harvesters possess the unique ability to engage with, open, and decipher any file type, encompassing intricate machine-generated data from a diverse array of sources. Leveraging embedded metadata from various data sources positions Metadata-Hub as an indispensable catalyst for propelling advanced analytics, elevating reporting standards, enhancing AI applications, and refining data orchestration efforts. It constructs a cohesive metadata warehouse, enabling effortless searching, sorting, and provisioning of metadata to an array of analytics, AI, reporting, and other tools, ensuring an uninterrupted flow of information.

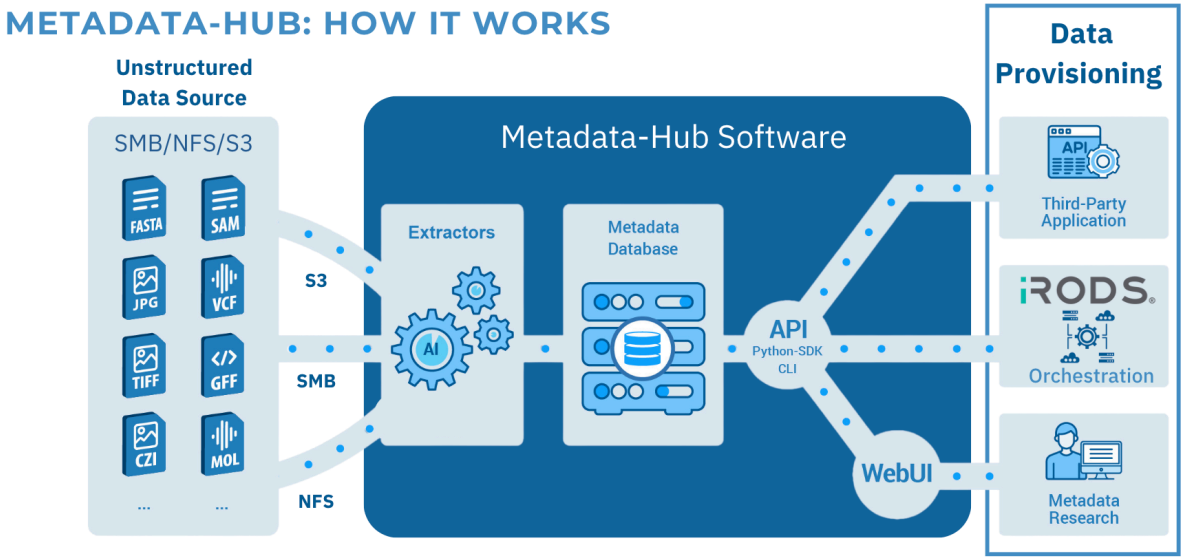

Figure 1. Metadata-Hub feeds metadata to applications and iRODS.

Embedded Metadata: The Core of Computational Efficiency

Distinct from conventional data, machine-generated files are endowed with extensive embedded metadata that fundamentally encapsulates the data's intrinsic value for computational applications. For instance, genomics FASTA files might encompass thousands of metadata tags, offering exhaustive insights essential for the majority of

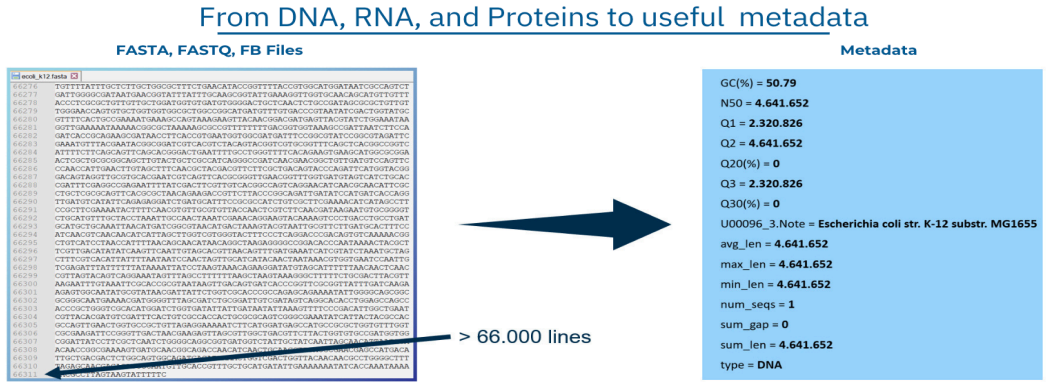

Figure 2. Example of machine-generated data: (left) genomics file and (right) harvested embedded metadata.

computational analyses without necessitating access to the full file. By prioritizing the extraction of this pivotal metadata, Metadata-Hub guarantees that crucial data elements remain readily accessible for searching, sorting, computation, analytics, and AI, substantially reducing the dependency on direct file access.

Metadata-Driven Workflows

The proactive movement of all machine-generated files to archival storage immediately after capturing their embedded metadata via Metadata-Hub presents a cost-effective strategy for managing storage while ensuring data remains available for computational applications. This methodology encompasses:

1. Prompt Metadata Extraction: Upon the creation or reception of a machine-generated file, Metadata-Hub swiftly extracts its embedded metadata, capturing essential details like data type, context, and additional pertinent information provided by the generating application.

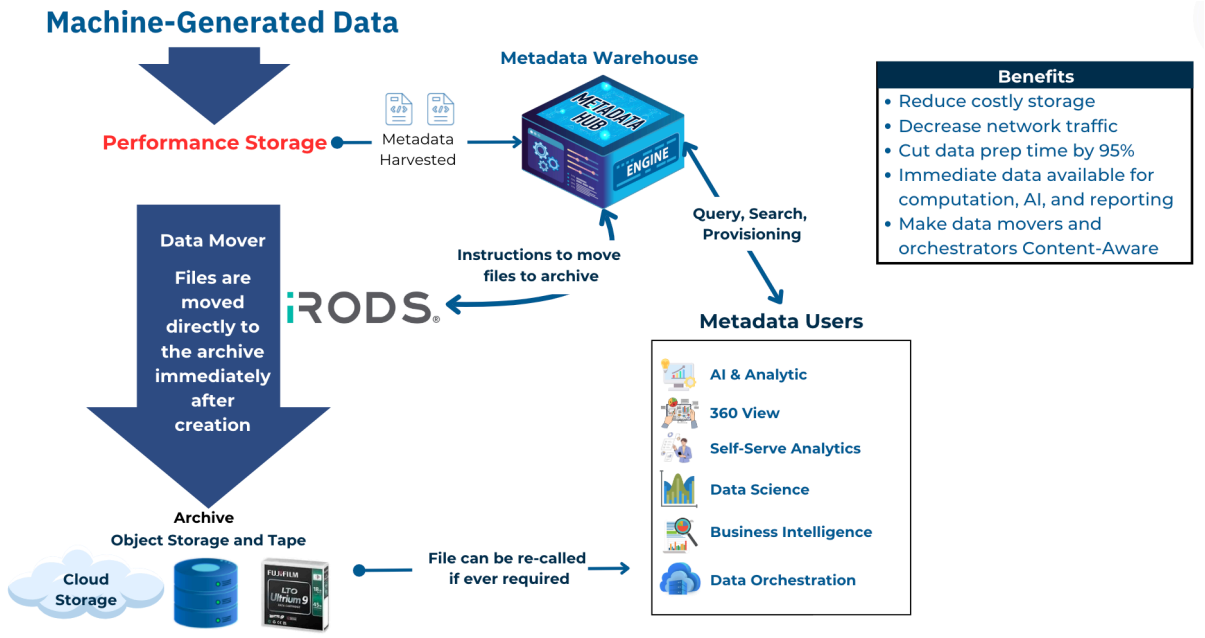

Figure 3. Embedded metadata is harvested from files on performance storage to feed metadata users.

2. Efficient Metadata Storage and Indexing: The extracted metadata is meticulously stored and indexed within Metadata-Hub, safeguarding the data's essence and context for accessible retrieval without the necessity for direct file interaction.

3. Strategic File Archiving: Subsequent to metadata extraction, the original file is relocated to archival storage, whether on-premises or cloud-based, offering a cost-efficient alternative to conventional high-performance storage systems.

4. Metadata Mesh: With pivotal data now encapsulated within Metadata-Hub as metadata, computational applications, analytics, and AI models can seamlessly query Metadata-Hub for the information required. This paradigm shift significantly accelerates data preparation and analytical phases, as these applications can directly engage with metadata rather than the full-scale processing of entire files.

5. Conditional File Recall: In scenarios where comprehensive analysis or specific demands necessitate access to the original raw file, such files can be efficiently recalled from the archival storage. Anticipated as an exceptional rather than routine requirement, this approach markedly diminishes the frequency of archive retrievals, further optimizing costs and enhancing operational efficiency.

6. Maximized Storage Efficiency and Cost Reduction: Archiving the majority of machine-generated files and leveraging metadata for computational objectives enables organizations to drastically curtail the volume of data

retained on costly, high-performance storage tiers. This strategy yields significant financial savings, elevates data management efficiency, and streamlines the utilization of voluminous machine-generated data for insightful decision-making.

Implementing Metadata-Hub to secure embedded metadata from machine-generated files and subsequently relegating those files to cost-conscious archival storage outlines a visionary approach to effectively managing escalating data volumes. This strategy ensures that data remains accessible for analytical and computational pursuits and optimally priced for storage and infrastructure management.

Advantages of a Metadata-Focused Architecture

- Substantial Storage Cost Savings: Facilitating the swift archival of raw data files to more affordable storage solutions immediately following metadata extraction permits organizations to realize considerable reductions in storage expenses. This strategic adjustment guarantees data accessibility without incurring the substantial costs associated with primary storage options.
- Accelerated Data Preparation: The methodology advocated by Metadata-Hub slashes data preparation durations by over 95%, smoothing the transition from data acquisition to insight derivation. This efficiency gain is achieved by rendering detailed metadata directly amenable for analysis, circumventing the laborious and time-consuming endeavor of retrieving raw data for preparation.
- Decentralized Architectural Model: Deploying Metadata-Hub as Docker containers proximate to data storage mitigates unnecessary data transfers and minimizes network strain. Capable of managing tens of billions of files, its scalable and decentralized architecture furnishes a unified, global perspective of metadata, augmenting query precision and streamlining metadata provisioning to diverse applications and data lakes.
- Robust Data Governance and Regulatory Compliance: Metadata-Hub's comprehensive provision of insights regarding data lineage, origination, and transformations empowers organizations to exert rigorous control over their data assets, aligning with both regulatory mandates and internal policy frameworks.
- Empowerment of Analytical and AI Initiatives: With metadata readily accessible in a structured, queryable format, Metadata-Hub dynamically formulates metadata marts specifically tailored to meet the nuanced demands of analytical and AI projects. This functionality ensures that data scientists and analysts have immediate access to exact data elements requisite for their endeavors, fostering innovation and expediting discovery.
- Enrichment of Data Lakes, Meshes, and Fabrics: The integration of Metadata-Hub into data lakes broadens the scope of data operations, encompassing enhanced reporting and more profound analytical tasks. This enhancement bolsters data mesh and data fabric architectures, augmenting data accessibility, interoperability, and utility across the organizational spectrum. By supplying embedded metadata—meticulously captured and organized by Metadata-Hub—to these frameworks, organizations can enrich their data lakes with intricate contextual details, elevating data discoverability, usability, and analytical depth.

Infrastructure Integration

Incorporating Metadata-Hub into existing storage infrastructures seamlessly amplifies data management capabilities without disrupting established systems. Its versatile architecture enables compatibility with an extensive array of storage solutions, ranging from traditional on-premises configurations to cutting-edge cloud-based platforms. This adaptability ensures that organizations can exploit the full potential of embedded metadata throughout their entire data ecosystem, irrespective of the underlying storage technology. Augmenting current environments with Metadata-Hub unlocks the complete potential of data assets, transforming every bit of information into a powerful tool for operational and analytical excellence. This integration not only simplifies data management practices but also maximizes the value extracted from each piece of unstructured data, driving forward innovation and operational efficiency.

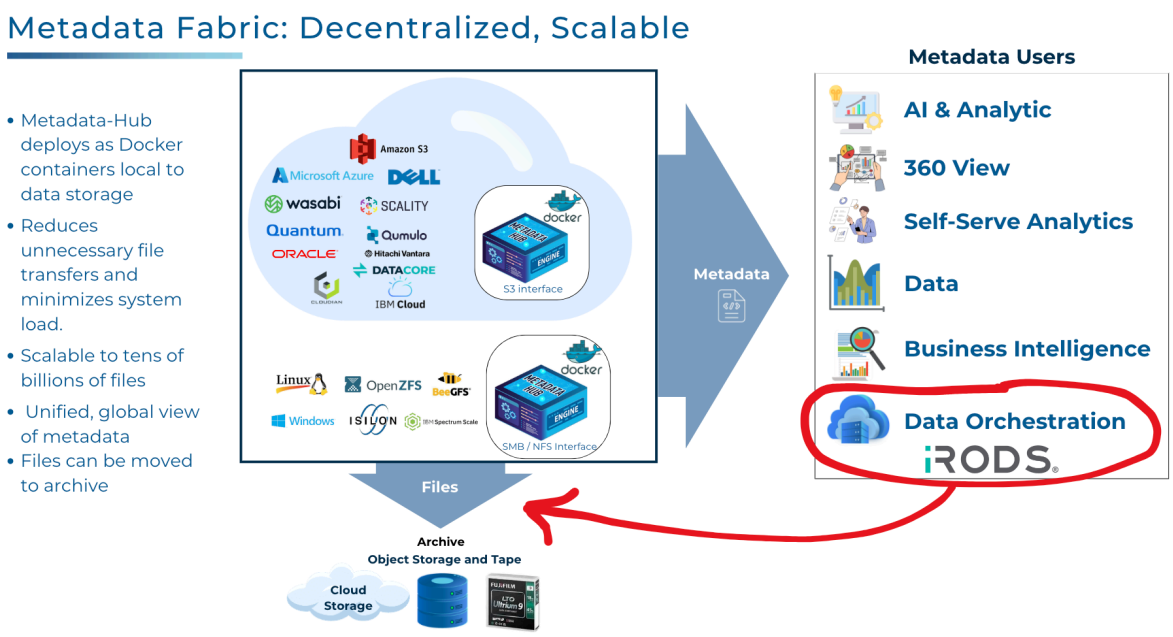

Figure 4. Metadata-Hub deploys locally to storage (cloud, data center, edge), and metadata is fed to iRODS.

Enhanced Data Mobility

The integration of Metadata-Hub with iRODS fundamentally enhances iRODS by making its data orchestration system contextually aware. This synergy allows for a more intelligent and precise management of data, informed by the rich insights of embedded metadata extracted by Metadata-Hub. As a result, iRODS can perform actions such as archiving, replicating, or retrieving data based on the content's relevance and context, not merely on basic file characteristics. This level of contextual understanding streamlines data workflows, optimizes storage management, and ensures the strategic alignment of data operations with organizational objectives and compliance demands. The collaboration between Metadata-Hub and iRODS represents a transformative shift towards a smarter, automated, and more nuanced approach to data management, enabling organizations to exploit their data assets with unprecedented efficiency and effectiveness fully. Here are the pivotal advantages of this integration:

- Superior Metadata Utilization: Metadata-Hub enables iRODS to employ intricate, contextually rich metadata to craft more refined data management policies and operations.
- Augmented Data Discovery and Accessibility: This collaboration facilitates more precise and efficient data discovery and accessibility. Users can effortlessly identify and leverage data based on comprehensive metadata, including content-specific nuances, which Metadata-Hub extracts and catalogs and iRODS efficiently administers.
- Automated Data Management Workflows: The integration of Metadata-Hub's capabilities to capture and classify detailed metadata at data creation or ingestion empowers iRODS to automate and refine data workflows. This encompasses automated archiving, replication, and distribution of data, rendering the entire data management lifecycle more efficient and less cumbersome.
- Strengthened Data Governance and Compliance: The enriched metadata management facilitated by Metadata-Hub equips iRODS with the tools to enforce more stringent data governance and compliance standards. Organizations can execute detailed policies and manage the data lifecycle with precision, ensuring adherence to regulatory standards and internal governance protocols.
- Cost Optimization and Storage Efficiency: By collaborating, Metadata-Hub and iRODS assist organizations in reducing their reliance on expensive storage solutions by enabling effective data archiving strategies based on deep metadata insights. This strategic approach leads to considerable savings in storage costs and enhances overall storage management and efficiency.

- Elevated Analytical and AI Readiness: The readily available, detailed metadata significantly shortens the data preparation time for analytics and AI, enabling immediate access to necessary data elements for data-driven decision-making processes. This readiness fosters a culture of rapid insight generation and accelerates the pace of innovation.

CONCLUSION

Incorporating Metadata-Hub into the data management strategy represents a transformative shift toward a more metadata-centric approach to handling the ever-growing volumes of machine-generated and unstructured data. By emphasizing the extraction and utilization of embedded metadata, Metadata-Hub redefines how organizations store, access, and analyze their data, leading to substantial improvements in operational efficiency, cost savings, and decision-making capabilities. The integration with iRODS further enhances this approach, offering a comprehensive solution that addresses the complexities of modern data management. Together, Metadata-Hub and iRODS provide a robust framework for organizations seeking to optimize their data storage strategies and unlock the full potential of their data assets in a cost-effective, efficient, and scalable manner. This synergy not only solves the immediate challenges of data management but also lays the foundation for future advancements in data utilization and analysis, ensuring organizations are well-positioned to thrive in the data-intensive era.

RSpace - iRODS integration: Update and next steps

Rory Macneil, Tilo Mathes
Research Space

Ander Astudillo
SURF

ABSTRACT

At the 2023 UGM the Stage 1 integration, which ensures that when external files linked to RSpace documents change location, the link integrity is maintained, solving the 'broken links' problem and enhancing the reliability of the research record maintained in RSpace, was presented. This year Stage 2 of the integration, which enables export of datasets and associated metadata from RSpace to iRODS, will be presented. This will open the possibility of association of data from RSpace with the potentially vast range of data – e.g. all the data tracked by iRODS and captured in a university's data storage facilities or all the data in a multi-university or a national project such as a biodiversity initiative. The RSpace data in iRODS will be discoverable via the associated metadata, enabling meaningful association with related data from other research resources, and re-use and repurposing of the data via, e.g., AI engines.

We will also introduce an idea for a Stage 3 integration that will enable RSpace to become a 'research data management front end' for iRODS. The working idea for this concept is as follows: The RSpace Gallery would act as a front end for iRODS to allow upload of data to iRODS (possibly even defaulting to storage in iRODS when files are added to the Gallery). Previewing/inline viewing of certain data types from iRODS should be possible, for example for image files (mirroring the current functionality for files stored in the RSpace Gallery).

Finally, in order to allow retrieval of data from iRODS (outside of RSpace), RSpace should be able to update a specific metadata field in the iRODS catalogue when files are linked to experiment entries in RSpace, so that querying the metadata catalogue for a particular document identifier could retrieve all data associated with that document (similar to seeing the linked documents in RSpace itself).

iRODS UGM 2024, May 28-31, 2024, Amsterdam, Netherlands
Author(s) retain copyright.

iRODS S3 API v0.2.0 with Multipart

Justin James
iRODS Consortium

ABSTRACT

The new iRODS S3 API can now present iRODS via the S3 protocol. This talk will share details about the first two releases, the implementation of various endpoints, and the state of Multipart transfers.

iRODS UGM 2024, May 28-31, 2024, Amsterdam, Netherlands
Author(s) retain copyright.

Streamlining iRODS: Kafka-based Data Pipelines

Peter Verraedt, Jo Wijnant
KU Leuven

ABSTRACT

We sketch how changes in the iRODS catalog can be captured in realtime into Apache Kafka, and how we materialize those master-records as a readonly 'view' in OpenSearch using Apache Flink stream processor. We show that the same tools can be used to continuously monitor project usage within iRODS. As a last application, we sketch how audit logs can be collected and processed, and show future challenges.

iRODS UGM 2024, May 28-31, 2024, Amsterdam, Netherlands
Author(s) retain copyright.

DAViDD: Initial data management solution for UNC's READDI AViDD Center

Terrell Russell
iRODS Consortium

ABSTRACT

The GA4GH Data Repository Service (DRS) standard is part of a family of standards for distributed, federated data analysis. Using standard workflow languages such as WDL, CWL, and Nextflow, these standards allow workflows to dispatch containerized tasks to run at appropriate locations, including across cloud providers and on-prem compute environments. The DRS standard provides an abstraction over distributed data sources, allowing these workflow tasks to authorize data access and access underlying data sets.

A DRS implementation over iRODS allows the iRODS data grid to expose data to this federated analysis ecosystem. The Federated Analysis System Project (FASP) components represent a formalization of the iRODS 'compute to data' pattern for the important Genomics and Health community.

iRODS UGM 2024, May 28-31, 2024, Amsterdam, Netherlands
Author(s) retain copyright.

The LEXIS Platform V2 and using iRODS for large scale data management

Martin Golasowski
IT4Innovations / VSB-TU Ostrava

Mohamad Hayek
Leibniz Supercomputing Centre

ABSTRACT

The LEXIS Platform allows easy access to HPC and Cloud resources with complex workflow orchestration and distributed data management based on iRODS. In our talk we present the improved version of the LEXIS Platform, including centralised metadata index, direct staging from iRODS zones and application container support. Extension of the LEXIS data management features are planned within the Horizon Europe EXA4MIND project, which focuses on handling large amounts of data between various data sources like databases or object storage and HPC infrastructure, including data publication features following FAIR principles. We also describe various use-cases for iRODS in the HPC environment, including using iRODS as data transfer gateway and preservation service based on EUDAT B2SAFE and testing of new iRODS interfaces such as S3 and HTTP API.

iRODS UGM 2024, May 28-31, 2024, Amsterdam, Netherlands
Author(s) retain copyright.

A Rust Library Crate for iRODS

Phillip Davis
Appalachian State University

ABSTRACT

Rust has a number of features that make it desirable for an iRODS client. For example, its strong type system and the borrow checker provide compile-time guarantees against unexpected runtime errors. I present an iRODS client library written in Rust. Connections are presently coupled with their connection pool, implemented using the deadpool library, but could be used independently with slight additions. However, deadpool allows connections to make limited use of async. Connection pools are generic over iRODS protocol encoding (currently only XML is supported), transport protocol, and authentication strategy. Message serialization and deserialization will only allocate heap memory if users explicitly opt in by using message types which own their data. Finally, TLS/SSL is delegated to operating system implementations using the native-tls library.

iRODS UGM 2024, May 28-31, 2024, Amsterdam, Netherlands
Author(s) retain copyright.

Integration of iRODS in a Federated IT Service through HTTP and Python API

Gautier Debaecker, Mathia Pagani
CC-IN2P3

ABSTRACT

The Federated IT Service (FITS) project, a collaborative endeavor between the IN2P3 (Institut national de physique nucléaire et de physique des particules) computing center and French national HPC Center named IDRIS (Institut du développement et des ressources en informatique scientifique), addresses the challenge of managing the escalating data volumes generated by research infrastructures. The project aims to consolidate computing and storage resources while maintaining control over hosting expenses and minimizing the ecological footprint of digital technologies.

Within the FITS project, iRODS was selected as the storage pooling solution, leveraging its established use within the IN2P3 Computing Centre. This implementation enables project users to seamlessly access their data without being aware of its physical location. The project outlines three primary access interfaces:

1. A 'simple' utilization through icommands for streamlined data sending and processing via the command line interface.

2. A python API-based graphical interface for local data management and processing.

3. A graphical/web interface utilizing HTTP API accessible through a user portal, facilitating user authentication through identity managers such as Indigo IAM or Keycloak. Initial functionalities will include data viewing and collection creation, with plans for expanded features such as group creation, iRODS rule configurations, and more in subsequent phases. While initial constraints of the HTTP API limit direct browser-based data transfers, the system will enable essential data interactions, marking the beginning of an evolving suite of functionalities for enhanced user accessibility and data management within the FITS project framework.

iRODS UGM 2024, May 28-31, 2024, Amsterdam, Netherlands
Author(s) retain copyright.

Update: The Intersection Between Policy-Based Data Management and Emerging Health Science Data Standards

Deep Patel, Mike Conway
NIH / NIEHS

ABSTRACT

iRODS is a domain-agnostic platform. The flexibility of iRODS flows from the fact that policy-based data management is at its core. By encoding metadata standards and management policies appropriate to a research domain into an iRODS data grid, an organization can create a powerful tool for data-driven research.

The Health Science domain is increasingly characterized by the co-location of data and compute, often 'bringing the compute to the data' and increasingly in a hybrid environment that spans cloud providers and bridges premises and cloud computing. This overview will look at GA4GH standards that intersect with the policy-based data management capabilities of iRODS for compute-to-data, access control, and federated search.

iRODS UGM 2024, May 28-31, 2024, Amsterdam, Netherlands
Author(s) retain copyright.

iRODS Metadata Templates Working Group: Building Blocks and Lessons Learned

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

ABSTRACT

With starts and stops over more than six years, the working group's efforts have spanned a number of ideas and projects. This paper covers that history, the group's decision-making, and some reusable code.

Keywords

iRODS, metadata templates, json schema, working group

PRE-HISTORY

The iRODS Consortium, founded in 2013 by the Renaissance Computing Institute (RENCI) at the University of North Carolina at Chapel Hill, mentions in its bylaws that "other committees and working groups may be established in pursuit of Consortium objectives." From 2014-2016, there was interest in the iRODS community around how to "do metadata well". These conversations were informal and led to the realization that a number of projects in the iRODS ecosystem were duplicating effort and working to solve some of the same issues. It was also noticed that each of these projects were trying to "boil the ocean" and solve a very hard problem, independently. A consensus formed that there was a better way forward.

The project first to appear was EMC's Metalnx, in 2015 [1]. Metalnx is a web GUI with a mission to help EMC sell more Isilon, its NAS storage platform at the time. EMC was interested in providing a templating engine where customers would enter and manage consistent metadata about the data stored in iRODS, making it more findable later.

Other projects at the time with similar goals around metadata included the Consortium's Cloud Browser [2], Utrecht University's Yoda [3], the DataHub from Maastricht University [4], the Dataverse project from Harvard and UNC-Chapel Hill [5], and the CyVerse project at the University of Arizona [6]. Each of these were solving some part or parts of this complicated puzzle.

The Metadata Templates Working Group [7] was formed in mid-2018 with the following motivating statement: "iRODS needs to help curators define and validate "good" metadata for their pipelines and environments." This was enough to schedule regular meetings and have a place and time to discuss, and perhaps consolidate, the various attempts to "do metadata well".

The Metalnx project was the farthest ahead at the time of the forming of the working group. Its initial designs had already generated mockups of how future development would provide the functionality and how it would fit within the existing layouts and framework of the codebase. Figure 1 shows an early rendering of the creation of a metadata template within Metalnx, as embedded in "The Junction" web portal of the National Institute of Environmental Health Sciences (NIEHS). Figure 2 shows how this metadata template would be rendered when presented to a staffer

iRODS UGM 2024 May 28-31, 2024, Amsterdam, Netherlands
[Authors retain copyright.]

tasked with filling out the required metadata. This work was done by Steve Worth (EMC) and Mike Conway (RENCI, and later, NIEHS).

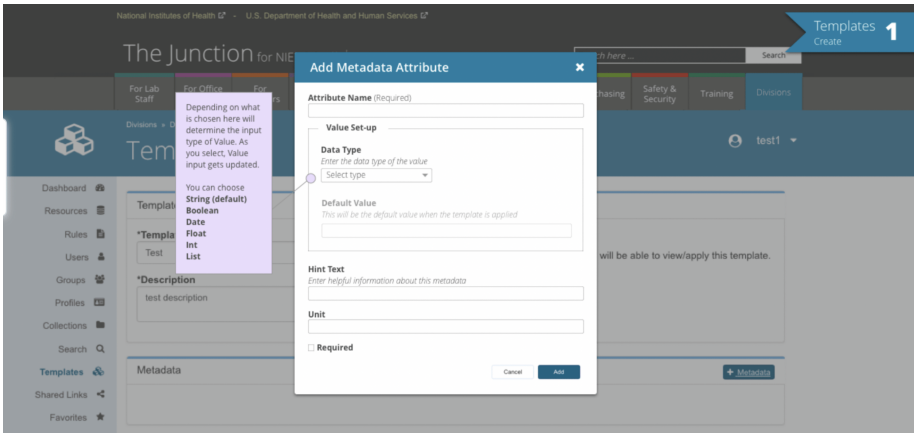

Figure 1. Early 2018 mockup of how MetaInx would prompt for the creation of metadata fields. This included defining the data type, help text, and hints for a default value and whether this field is required.

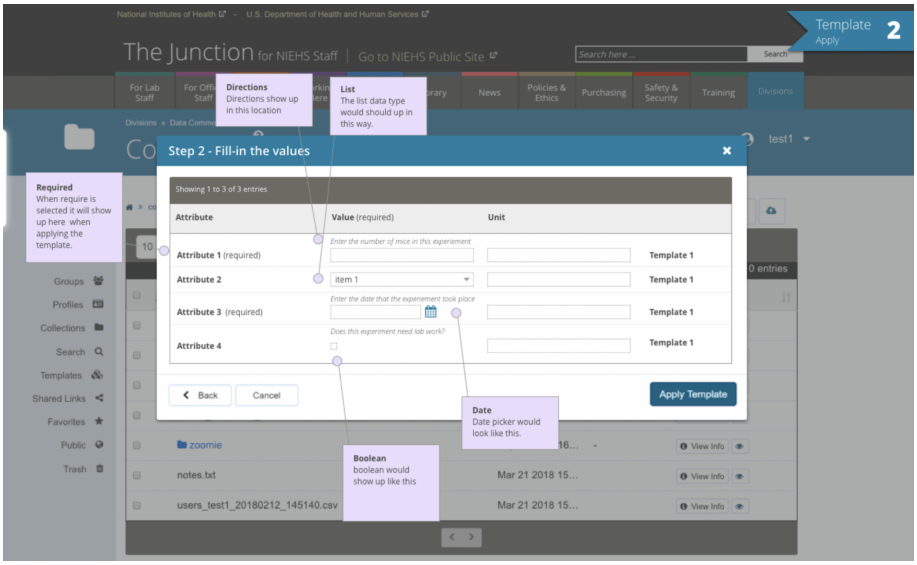

Figure 2. Early 2018 mockup of how MetaInx would prompt for the filling in of a metadata template including multiple fields, some required.

The Cloud Browser, from the iRODS Consortium, had also worked on how to formalize metadata being stored within iRODS. Independent from others, JSON Schema had been selected to define a template, allowing for validation to be applied and well-formed-ness to be guaranteed. These templates defined a schema to which the metadata on an object or collection must conform. These schema files were stored at the root level of the Zone in an ".irods" collection, available to all users. This work involved the creation and maintenance of a parser (for the templates, and the data), a validator (to compare the data to the schema(s)), a resolver (for determining which schemas must be applied and

validated), and an exporter (for rendering the results of the resolved and validated metadata). This code handled the combining and merging of multiple templates into a java object. This work was done by Mike Conway (RENCI, and later NIEHS), Cesar Garde (RENCI), and Terrell Russell (RENCI).

These two projects required defining some of the endpoints to be used for metadata templates in both the web client applications as well as Jargon, the iRODS Java client library [8]. This led to a discussion about a Swagger API (later known as OpenAPI), which in turn, led to more conversation about a new REST API for all of iRODS which would include additional metadata template endpoints.

In May 2017, Rick Skarbez (RENCI), gave a TRiRODS presentation [9] on his work showing a full double roundtrip flow (Figure 3) from client to server where a new file is uploaded, metadata templates are applied, the user populates the rendered template, and then that metadata is validated and saved.

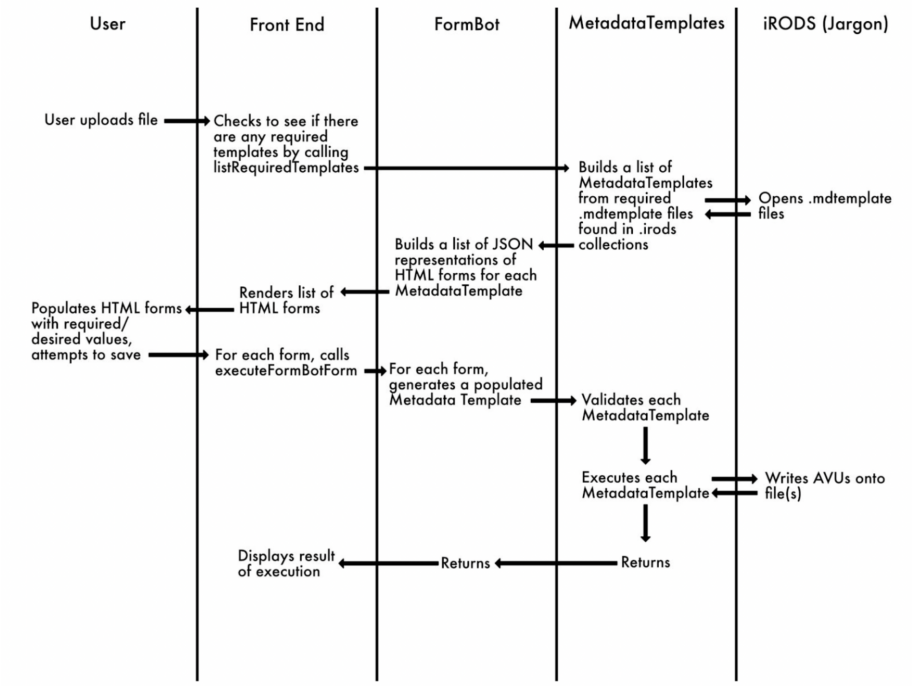

Figure 3. From May 2017 TRiRODS: A swimlane representation of a full double roundtrip from client to server illustrating template creation, population, and validation.

And so, in June 2018, the Metadata Templates Working Group was formed and by March 2019, the Metalnx metadata templates were stored in the Metalnx database as jsonschema. This was internally consistent, but didn't allow other applications to see or use these templates.

2019

By June 2019, Maastricht and Utrecht demonstrated iRODS rules to provide a round trip from JSON to AVU to JSON [10, 11]. This was important as other tools were using JSON to represent their data models, but iRODS has its metadata stored in AVUs in the catalog (available to the iRODS API). This solution handled type information for each element and nesting (hierarchical metadata) through the use of the unit field. It handled metadata for both data objects and collections. This work was done by Paul van Schayck (Maastricht), Ton Smeele (Utrecht), Daniel

Theunissen (Maastricht), and Lazlo Westerhof (Utrecht).

At this time, there were multiple non-Consortium implementations appearing, and they were showing signs of convergence. These included Yoda from Utrecht, DataHub from Maastricht, the Data Commons from NIEHS, and CyVerse from the University of Arizona.

In addition to the working group's efforts, Stanford's Center for Expanded Data Annotation and Retrieval (CEDAR) [12] had come online and was becoming a familiar interface and home for an editor to define and save metadata templates as schemas.

At this point (iRODS User Group Meeting, June 2019), it had become clear there were a number of moving parts that would require continuous maintenance and engagement, and that many of them were outside the scope of the working group's mandate.

The working group identified six main elements of the work up to this point.

1. Definition / Representation of the Schema (CEDAR itself?, NIEHS)
2. Tools for template/schema creation / curation / versions / management (CEDAR itself?, NIEHS)
3. Tools for managing the data with relation to the templates (DataHub+, Yoda)
4. Translation from schema to AVUs and back (DataHub+, Yoda)
5. Multiple UIs / utilities handling the translation/presentation (Yoda, NIEHS)
6. (saved) Search queries and results, virtual collections

A month later, upon some reflection and reordering, this list of six was refined to five (Figure 4).

Layer	Functionality	Implementation	Implementor(s)
5	Multiple UIs / Utilities handling the translation/presentation	Yoda, Metalnx	Yoda, NIEHS
4	Tools/API for translation from template to AVUs and back	JSON<->AVU	DataHub+, Yoda
3	Tools for managing the AVUs with relation to the templates	rules/policy	DataHub+, Yoda
2	Tools for template creation / curation / versioning / management	CEDAR	CEDAR
1	Definition / Representation of a Template	JSON Schema	JSON Schema Organization

Figure 4. The five ordered layers of Metadata Templates functionality along with their 2019 implementations.

Later that fall, the Yoda project at Utrecht demonstrated the first use of an external schema applied to iRODS AVUs. It was also then that discussion honed in on the identification that atomic application of AVUs at the database level is more important than batch processing of multiple requests to update.

In October of 2019, the working group had identified several operations within a Swagger API:

1. Resolve MTs based on an object/collection
2. List attached MTs on an object/collection
3. Attach/Apply MT to an object/collection as required/optional
4. Remove MT from an object/collection

5. List overall available MT in the pool
6. Resolve JSON schema(s) that defines the metadata to be applied via template X to collection Y
7. POSSIBLE - Rasterize? Set of nested/attached schemas down into a single schema

These operations showed the various actions that could be taken within the context of metadata templates in iRODS.

2020

Early the next year, in February 2020, focus turned on where these schemas could be managed. The group worked on the creation of 3-4 CEDAR-based JSON schemas for testing as well as evaluating the use of CEDAR as schema editor before exporting those schemas to be used in local evaluation within iRODS. This period of time also led to the discovery that it would probably be best to use API PEPs rather than database PEPs for defining the programmatic hooks to evaluate the metadata and application of metadata templates.

April 2020 saw the addition of atomic AVUs API endpoints to iRODS. This work was spearheaded by the working group's necessity to edit multiple AVUs without leaving a data object or collection in an invalid state as compared with an applied template.

In July of 2020, the working group decided that CEDAR was a good editor, but would not be suitable as publisher or host of any created schemas. The hosting was not free and required API access for live interactions. The hosting would have to be elsewhere for an open system. It was also at this time that the investigation of jsonforms [13] evaluated that building xml/html forms from schemas was still a hard problem and no library solution was yet available we were willing to build upon.

In August 2020, the Yoda project at Utrecht announced an atomic endpoint and discussion quickly moved to the possibility of aggregating templates recursively. Aggregation leads to possible incompatible collisions of identically named attributes and differently defined enumerations.

However, the working group defined the elements of architecture for the first time as the layers continued to be separated and clarified.

1. CREATION/DEFINITION of templates (punt to CEDAR / others)
2. HOSTING of templates (perhaps CEDAR, perhaps irods.org or github)
3. BINDING/MANAGEMENT of templates to collections/data (part of MTWG MVP)
4. USE of templates in GUI (part of MTWG MVP)

This ownership also clarified some relevant API components for the group to focus on (as well as what NOT to focus on):

1. CLIENT/BROWSER: some javascript code to execute client side, wraps an Ajax POST call to the web server
2. WEB SERVER: passes the rule call onto iRODS
3. iRODS PYTHON RULE ENGINE: processes the API call

By November of 2020, CEDAR had announced their move to JSON-LD, which did not yet have sufficient library support in the ecosystem for the working group to also adopt it.

2021

January 2021 saw some additional clarity in the ecosystem as GO FAIR [14] had decided to use Jinja templates for their rendering and layout. This led the working group to step back and assess that there is no ”one ring” to rule them all. Different applications will continue to handle rendering themselves, and that the tools to do so will continue to move quickly. The working group should do a good job on the API to define the standard that others can assume will work. Various GUIs can ask for templates, render them however they may see fit for their purpose, and then send back filled information. It is not the role of the API to be in the ”application space”.

The next month saw general agreement on this point and the decision was made to be schema and application agnostic. The JSON Schema standard was enough to evaluate whether a schema was well-formed, and then whether any associated metadata was valid against any particular schema. These were the building blocks that the working group should work towards.

After the second virtual iRODS User Group Meeting, in June 2021, this earlier finding was made more explicit. Subject areas should, and will continue to, drive the work of defining schemas for their respective areas of expertise. iRODS should not define or manage templates for anyone. iRODS should validate and get out of the way.

2022

In February 2022, attendance at eResearchNZ [15] found many ongoing topics and discussions validated and echoed the working group’s efforts. Curators want to be in charge of what is required for quality datasets to be annotated well. They want to define the parameters and then let the computers enforce and flag for humans to come help, an archival and systems perspective on human-in-the-loop.

By March 2022, KU Leuven had begun building a portal (which would become ManGO [16]) including templates, an editor, handling both required and optional metadata on both collections and data objects.

In June it was decided a paper like this one should be written. Eventually, it would happen, but not for a couple more years.

It was clear by August 2022 that the community was ”ahead” of the Consortium and would continue to iterate more quickly where the use cases demanded attention. They were building user-facing code and handling edge cases to solve today’s needs. The iRODS server should be providing building blocks for the client code to depend on. The community was also leaning into Python for prototyping and quick iteration and planning to move to C++ later once the patterns and inputs/outputs were agreed to and good.

Near the end of 2022, the Minimum Information About a Microarray Experiment (MIAME) paper was reviewed [17], providing an example of a scientific discipline inventing their own minimal standard for interoperability. Other work at the time was beginning to demand machine actionable data management plans. The working group continued to see these efforts as continued validation. iRODS should be a consumer of these efforts and facilitate easier conversations across disciplines.

2023

March of 2023 brought more validation from the Research Data Alliance’s 20th Plenary Meeting (RDA20) in Sweden [18]. This meeting showed a similar struggle to find consensus. Different disciplines have their own language and details that matter to them. Getting consistency across groups is very hard, if not impossible, since there are so many competing interests and not enough incentive to work with every possible collaborator.

KU Leuven had made progress as well. They produced and shared a schema editor written in Javascript. They had developed a process to handle template versioning and thought through what happens when a new template is applied to old data. Their templates were namespaced, which prevented collisions, and were based on project-level management of these multiple templates.

July 2023 required more discussion to think about how these different implementations might be consolidated. It was considered whether iRODS should store the templates, perhaps in a new database table in the iRODS Catalog. It was shared that KU Leuven was using a template to render forms for the user to complete, but not using it to validate the metadata inserted into those forms. This suggested two different types of template might need to be considered - one for the form information, and one for the metadata itself. IT4I had chosen to implement a single minimal schema which was used to enforce a clean export of data to elasticsearch for consolidated searching across multiple iRODS zones. This avoids the collisions and namespacing issues faced by others, but of course, was limited in that the single schema had to account for everything important across multiple projects and systems.

It was at this time four actions came into focus, to be considered as iRODS microservices that needed to be implemented with JSON Schema:

1. Attach (type, schema, object_id) - This would take three parameters to define the type of schema this is, the reference to the schema, and the iRODS catalog object this schema is to be attached to. The type could include "url" (and "schema" would store a string), "irods_schema" (and "schema" would store an id from the new schema table), or "form" (where "schema" would store KU Leuven wrapper or form information).
2. Detach (type, schema, object_id) - This would simply remove the attachment.
3. Export/Collapse/Rasterize/Gather/Dump (object_id, recursive) - This would find all associated schemas on the specified iRODS object and construct an "effective" schema. If this was called recursively, it would include all schemas attached to parent objects up to the root of the logical namespace.
4. Validate (object_id, recursive) - This would run gather (above) to build the effective JSON Schema for an object, get and build a JSON payload with the current AVUs of the same object, and then run the payload through the effective schema and return the result.

The next month saw Utrecht declare that their implementation had separated the different elements of the Yoda system to be more clear about where metadata templates were defined and where they were enforced. Their UI rendering, including forms, was handled entirely by React. Their metadata schemas were defined in JSON Schema. The research space where users were doing live filesystem manipulations and active work had no metadata template validation or enforcement. The vault space in Utrecht had requirements that had to be met and, therefore, were having the JSON Schema templates being enforced on its metadata. This raised the question of whether the schema information in iRODS needed to be protected by the metadata_guard plugin. Disallowing users to manipulate the schemas applied to their data must be enforced.

In November 2023, the iRODS Consortium had an initial implementation of the first two actions. Attach and Detach were working, but needed robust error checking and to return good error messages to the caller. Gather and Validate were next on the list.

2024

The new year brought some practical questions that were unforeseen before there was running code. The working group discussed multiple schemas being attached to the same collection and whether Gather should use "AllOf" to combine the schemas or simply loop through all that were found. As soon as collection templates existed, the community wanted to see templates for users, groups, resources, and data objects as well. Could these be handled with the same code, or would it require specific implementations for each?

By May of 2024, the code had matured to a full implementation. Attach and Detach remained the same. Gather returned an array of attached schemas, rather than trying to consolidate or collapse them into a single combined schema (avoiding collisions and conflicts). It also handled the recursive flag. Validate could now handle evaluating a data object or a collection, which would validate all the data objects below the collection.

CONCLUSIONS

This working group discovered a number of findings to be recorded for posterity. We hope that these findings are helpful to the next set of practitioners, implementers, and standards committees.

Generally, there are a great number of metadata template scenarios that require site-specific or domain-specific knowledge with diverse interfaces for special cases. A general solution is not a reasonable expectation and would be too expensive to solve for and to maintain.

Separately, metadata template management, itself, is too wide a task for the dependent system that will be using those templates. In our case, the dependent system was the iRODS server and its policy enforcement.

iRODS should focus on the capabilities and functionality required rather than defining policy and/or the schemas for the domain-specific applications and users. iRODS cannot (or should not) be defining the templates for others. It should provide functionality only to validate these templates against the data in question. The templates should be managed (created, versioned, hosted) elsewhere.

iRODS should provide 70-80% of the original intent of the scope and scale of this working group. Then, the community will use, test, incorporate, and iterate on the prototype Python functions. Once these are vetted and in production, they can be rewritten in C++ to be more performant and to be included in the iRODS server as a built-in feature.

RUNNING CODE

The following code is shared to show the four functions in action - attach, detach, gather, and validate.

```
# attach a template
$ irule -r irods_rule_engine_plugin-irods_rule_language-instance \
    "metadata_templates_collection_attach('*logical_path', '*schema_location', 'url')" \
    '*logical_path=/tempZone/home/rods/thedir%*schema_location=\
    https://raw.githubusercontent.com/fge/sample-json-schemas/master/jsonrpc2.0/jsonrpc-request-2.0.json' \
    ruleExecOut


# show AVU
$ imeta ls -C thedir
AVUs defined for collection /tempZone/home/rods/thedir:
attribute: irods::metadata_templates
value: https://raw.githubusercontent.com/fge/sample-json-schemas/master/jsonrpc2.0/jsonrpc-request-2.0.json
units: url


# detach a template
$ irule -r irods_rule_engine_plugin-irods_rule_language-instance \
    "metadata_templates_collection_detach('*logical_path', '*schema_location', 'removeme')" \
    '*logical_path=/tempZone/home/rods/thedir%*schema_location=doesnotexist' \
    ruleExecOut


# gather, print to stdout
$ irule -r irods_rule_engine_plugin-irods_rule_language-instance \
    "metadata_templates_collection_gather('*logical_path', '*recursive', *schemas); \
    writeLine('stdout', *schemas)" \
    '*logical_path=/tempZone/home/rods/thedir%*recursive=0%*schemas=""' \
    ruleExecOut
```

```
# validate data object
$ irule -r irods_rule_engine_plugin-irods_rule_language-instance \
    "metadata_templates_collection_gather('*logical_path', '*recursive', *schemas); \
    metadata_templates_data_object_validate('*data_object_path', *schemas, *rc); \
    writeLine('stdout', *rc)" \
    '*logical_path=/tempZone/home/rods/thedir%*recursive=0%*schemas=""%\
    *data_object_path=/tempZone/home/rods/thedir/a.txt%*rc=""' \
    ruleExecOut


# validate a collection
$ irule -r irods_rule_engine_plugin-irods_rule_language-instance \
    "metadata_templates_collection_gather('*logical_path', '*recursive', *schemas); \
    metadata_templates_collection_validate('*logical_path', *schemas, *recursive, *errors); \
    writeLine('stdout', *errors)" \
    '*logical_path=/tempZone/home/rods/thedir%*recursive=0%*schemas=""%*errors=""' \
    ruleExecOut


$ bash bats-core/bin/bats test_metadata_templates.bats
test_metadata_templates.bats
✓ collection - attach, gather, detach template
✓ attach bad schema
✓ validate data object
✓ validate collection

4 tests, 0 failures
```

FUTURE PLANS

The iRODS Metadata Templates Working Group has nearly completed its work. Within the next year, the Consortium plans to ship these functions in the next major version of iRODS, as part of the Python Rule Engine Plugin (PREP). If there is interest and demand, they could be ported to C++ for speed.

REFERENCES

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

[2] iRODS Cloud Browser https://github.com/irods-contrib/irods-cloud-browser

[3] Yoda https://www.uu.nl/en/research/yoda

[4] DataHub+ https://datahubmaastricht.nl/

[5] Dataverse https://dataverse.org/

[6] CyVerse https://cyverse.org/

[7] iRODS Metadata Templates Working Group https://github.com/irods-contrib/irods_working_group_metadata_templates

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

[9] Skarbez, R. 2017. TRiRODS: Metadata Templates in iRODS https://www.youtube.com/watch?v=_b4AvhhG7mc

[10] JSON to AVU to JSON https://github.com/MaastrichtUniversity/irods_avu_json

[11] JSON to AVU to JSON ruleset https://github.com/MaastrichtUniversity/irods_avu_json-ruleset

[12] Musen MA, Bean CA, Cheung K-H, Dumontier M, Durante KA, Gevaert O, Gonzalez-Beltran A, Khatri P, Kleinstein SH, O'Connor MJ et al.. 2015. The Center for Expanded Data Annotation and Retrieval. Journal of the American Medical Informatics Association, JAMIA. 22 (6) 1148-1152 https://doi.org/10.1093/jamia/ocv048 https://metadatacenter.org/

[13] JSONForms https://github.com/eclipsesource/jsonforms

[14] GO FAIR https://www.go-fair.org/

[15] eResearch NZ 2022 https://www.reannz.co.nz/news-and-events/eresearch-nz-2022

[16] Borgermans, P., Montes, M., Barcena Roig, I. 2023. ManGO: A web portal and framework built on top of iRODS for active research data management https://github.com/kuleuven/mango-portal

[17] Brazma, A., Hingamp, P., Quackenbush, J., Sherlock, G., Spellman, P., Stoeckert, C., ..., Vingron, M. (2001). Minimum information about a microarray experiment (MIAME)—toward standards for microarray data. Nature genetics, 29(4), 365-371. https://doi.org/10.1038/ng1201-365

[18] Research Data Alliance (RDA) 20th Plenary. 2023. https://www.rd-alliance.org/plenary-core/rda-20th-plenary-meeting-gothenburg-hybrid/