iRODS Labels: A Proposal for a Complementary Labeling Mechanism in iRODS

A. Tsyganov
The University of Groningen,
Groningen, Netherlands
a.tsyganov@rug.nl,
rdms-support@rug.nl

ABSTRACT

iRODS is a well-known research data management system with a wide range of functionalities, including one of the most valuable components — its metadata engine, which allows scientists to associate data objects with descriptive attributes or tags. In iRODS, metadata are implemented as AVU (Attribute, Value, Unit) triples, with each instance attached to a single object. As a result, expressing relationships where one logical label or descriptor is shared across multiple objects requires duplicating metadata entries, rather than defining a single reusable entity linked to multiple objects.

This work proposes an approach that introduces labels — a complementary metadata construct that enables mapping a single label to multiple data objects. This mechanism coexists with the existing AVU-based metadata model and provides additional capabilities for data organization, cross-reference search, and metadata templating. This approach simplifies management of logically related data objects, reduces duplication of metadata key–value pairs, and improves consistency across large-scale collections.

The implementation includes an additional module in iCommands (ilabel) and requires minimal modifications to several components of the query processing subsystem to support label resolution within GenQuery and GenQuery2.

The system architecture and a proof-of-concept are presented, developed by building iRODS from source, extending the codebase with new components and database entities, and modifying the query processing layer. Parts of the implementation were assisted by an AI-based code generation tool.

Keywords

RUG RDMS, metadata templates, iRODS Labels, AVUs, iRODS compilation, python-irodsclient, ICAT database, ilabel.

INTRODUCTION

With the growth of scientific data catalogs over the last decades, one of the principal challenges of research data management is how to efficiently build metadata definitions and catalog data in different scientific areas. In particular, this includes requirements for storing experimental data in compliance with policies like FAIR [1] or GDPR [2][3]. With the rapid growth of scientific metadata over the last decade [4], accurate dataset description has become essential. iRODS (The Integrated Rule-Oriented Data System) [5] [6] [7] addresses this task.

One of iRODS's key functionalities is the metadata engine that stores descriptive metadata as AVU (Attribute, Value, Unit) triples [8]. AVUs are designed to serve as unique descriptors of a particular file or folder. This functionality helps to describe data on different levels. These can be general publication metadata or granular parameters of a particular experiment attached to its result [9]. Good tagging of data with metadata also simplifies data search. However, the existing iRODS metadata engine has one practical limitation — metadata duplication.

iRODS UGM 2026, June 29 – July 2, 2026, Barcelona, Spain

For instance, a scientific lab needs to mark all its experimental data with the parameters of the setup used, and only a fraction of those parameters are actually changing between runs. Then each raw dataset gets many identical AVUs. This helps with search but does not help with metadata management: for instance, a typo in a shared metadata value will require updating the metadata for all datasets.

It is also common practice to use metadata templates to describe particular types of data [10]. At present, there are thousands of available metadata templates across different scientific areas. Let us imagine that a data steward was using a version of the metadata template that requires a major update. Then all files and folders that rely on this metadata template structure need to be updated. The Research Data Management System [11] at the University of Groningen [12] uses metadata templates to tag relevant data and datasets for publication, and a specific design decision has been made on how to manage these templates. The solution uses a mix of custom Python service code with a special syntax and logic for the AVUs that are stored in iRODS.

ARCHITECTURE

In addition to the existing iRODS metadata AVU mechanism, complementary labeling is proposed. In comparison to the existing one-to-one architecture (fig. 1), which has its value for many use cases, the proposed independent data labeling introduces one-to-many referencing between metadata and objects (files, folders, resources) in iRODS (fig. 2) — iRODS Labels. The current implementation focuses on data objects (files) and collections (folders).

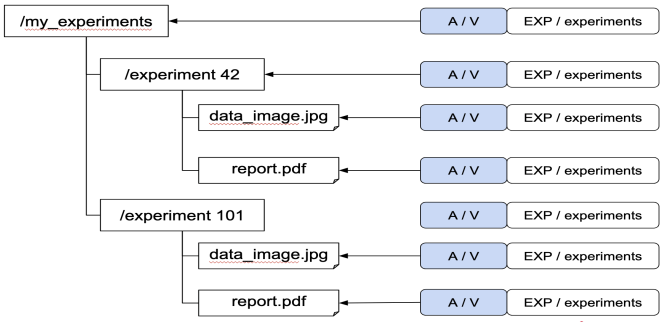

Figure 1: iRODS AVUs principal architecture example for data object and collections

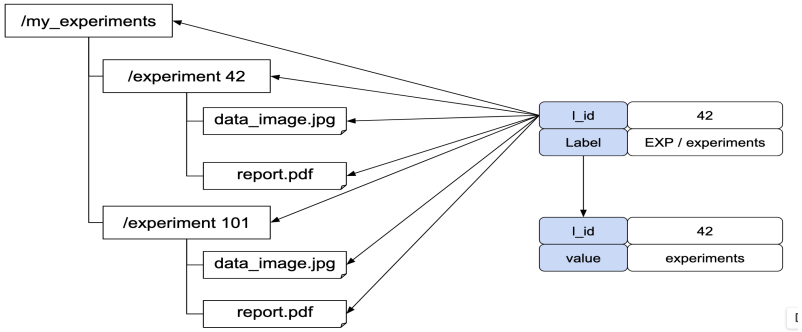

Figure 2: iRODS Labels principal architecture example for data object and collections

The design starts with the additional database tables (fig. 3). Being an additional functionality, iRODS Labels intends to introduce minimal changes to the core iRODS code and schema.

Figure 3: iRODS Labels database schema

The main table of the schema is *r_label* — label definitions. It contains the label name, visibility and parent. One of the architectural choices is to add labels to the tree structure. In the current implementation, child labels can have only one parent label; however, a potential future extension would allow child nodes to have multiple parent nodes. This would make labels truly suitable for implementing multiple metadata templates with minimal duplication.

The next table is *r_label_value* — one value payload per label. Names and values are split by design to allow different visibility patterns to be applied per user. For instance, the label itself can be publicly available but its value should not be visible to users because of privacy restrictions. The following workflow for the data manager then applies: all data in projects related to the genome study must be tagged with the "GEN" label, but only the data manager can see the values that the institution applies for this tagging. In general, the design assumes that the value of the label is a sort of JSON data and there is another field that is meant to hold a JSON Schema for data validation.

iRODS Labels has its own ACL implementation for label name and value visibility. These ACLs are stored in *r_label_access* — per-label ACL; user + access level (read/modify/own). In the *r_label* table there is a visibility flag. There are two values: public (visible and searchable by any user) and private — in which case the same ACL from *r_label_access* applies as for the label value. Then there are tables that build a connection between data objects/collections and labels:

- *r_data_label* — attaches labels to data objects
- *r_coll_label* — attaches labels to collections; includes the inherit flag. The inheritance flag dictates how iRODS applies labels to newly registered objects. If an object's parent collection has a label marked inherit=True, then the new object will receive the same label in r_coll_label with inherit=True.

This is the key architectural decision point in the implementation. The general iRODS behaviour is not to use foreign keys (FKs) in the database schema, but to resolve all such cascade operations inside db_plugin.cpp hooks. The described implementation follows the same pattern; however, it may be worth considering the introduction of schema FKs and ON DELETE CASCADE and ON INSERT triggers for easier consistency management in complex transactions — not only for labels but for standard AVUs and ACLs as well.

It should be noted that for the label value and validation fields, the TEXT data type is best suited for PostgreSQL; however, in Oracle the equivalent is the CLOB data type, and special syntax is required when the data size exceeds

4,000 characters. Thus, to be on the safe side, setting the data type to VARCHAR(4000) may be a simpler option for cross-database support.

DEVELOPMENT AND TEST ENVIRONMENT

The proof-of-concept was developed and tested in a virtualized environment using Vagrant with VirtualBox, running Ubuntu 22.04 LTS (ubuntu/jammy64). iRODS 4.3.5 was compiled from source, and the iRODS Labels plugin was built against that source tree. PostgreSQL was used as the iCAT database backend. The key components of the environment are:

- Ubuntu 22.04 LTS
- iRODS 4.3.5 (compiled from source)
- PostgreSQL
- Python & python-irodsclient
- Vagrant + VirtualBox

The iRODS source was patched at seven points before compilation (see Implementation). The plugin build and iRODS server configuration were fully automated via provisioning scripts, making the environment reproducible from a single vagrant up command.

The implementation of iRODS Labels was assisted by Anthropic's Claude AI, Sonnet 4.6 model [13].

IMPLEMENTATION

The following is a list of core iRODS files that need to be changed to implement the architecture described above:

- API Registration

  plugins/api/include/irods/plugins/api/api_plugin_number_data.h

  Registers API slot #20100 for the label RPC

- GenQuery v1 (iquest)

  lib/core/include/irods/rodsGenQueryNames.h

  Adds LABEL_* column name string constants

  plugins/database/src/general_query_setup.cpp

  Registers sTable / sColumn / sFklink entries for label tables

  plugins/database/src/general_query.cpp

  genqAppendAccessCheck() ~L1649: injects visibility ACL WHERE clause for private labels on every label-touching query

  server/icat/src/icatGeneralQuerySetup.cpp

  Wires label columns into the iCAT query engine setup

- GenQuery v2 (iquery)

  server/genquery2/include/irods/private/genquery2_table_column_mappings.hpp

  Maps label column IDs 11000–11032 to table.column

server/genquery2/src/genquery2_sql.cpp

Adds r_label join logic for v2 SQL generation*

- Catalog Lifecycle Hooks

  plugins/database/src/db_plugin.cpp

  Handles major data consistency intervention across the object lifecycle (create / delete / update).

The major code intervention into the iRODS core happens in the db_plugin.cpp file in the following functions: _delColl (line ~1058), chlRegDataObj (line ~2949), chlUnregDataObj (line ~3449), chlRegColl (line ~5000).

The major functional check required is that the general iRODS operation does not suffer from potential code errors in iRODS Labels functionality. This requires iRODS server testing with heavy parallel data processing. The main issue to check is that processes do not leave any unresolved locks and do not cause stale waiting connections.

The rest of the changes are straightforward and do not require much code review (Code. 1) (Code. 2) (Code. 3).

Code snippet 1: plugins/database/src/general_query_setup.cpp code additions

```
/* Labels plugin — table registrations */
sTable( "R_LABEL",       "R_LABEL",       0 );
sTable( "R_LABEL_VALUE", "R_LABEL_VALUE", 0 );
sTable( "R_DATA_LABEL",  "R_DATA_LABEL",  0 );  sTable( "R_COLL_LABEL",  "R_COLL_LABEL",  0 );
/* Labels plugin — column mappings (COL_LABEL_* defined in irods/label_ops.h) */
sColumn( COL_LABEL_ID,              "R_LABEL",       "label_id" );
sColumn( COL_LABEL_NAME,            "R_LABEL",       "label_name" );
sColumn( COL_LABEL_DESCRIPTION,     "R_LABEL",       "label_description" );
sColumn( COL_LABEL_CREATOR_ID,      "R_LABEL",       "creator_id" );
sColumn( COL_LABEL_CREATE_TS,       "R_LABEL",       "create_ts" );
sColumn( COL_LABEL_PARENT_ID,       "R_LABEL",       "parent_label_id" );
sColumn( COL_LABEL_VALUE,           "R_LABEL_VALUE", "value" );
sColumn( COL_LABEL_VALUE_MODIFY_TS, "R_LABEL_VALUE", "modify_ts" );
sColumn( COL_DATA_LABEL_DATA_ID,    "R_DATA_LABEL",  "data_id" );
sColumn( COL_DATA_LABEL_LABEL_ID,   "R_DATA_LABEL",  "label_id" );
sColumn( COL_DATA_LABEL_ASSIGN_TS,  "R_DATA_LABEL",  "assign_ts" );
sColumn( COL_COLL_LABEL_COLL_ID,    "R_COLL_LABEL",  "coll_id" );
sColumn( COL_COLL_LABEL_LABEL_ID,   "R_COLL_LABEL",  "label_id" );
sColumn( COL_COLL_LABEL_ASSIGN_TS,  "R_COLL_LABEL",  "assign_ts" );
```

Code snippet 2: irods-src/lib/core/include/irods/rodsGenQueryNames.h code additions:

```
{ COL_LABEL_ID,               "LABEL_ID"              },

{ COL_LABEL_NAME,             "LABEL_NAME"            },

{ COL_LABEL_DESCRIPTION,      "LABEL_DESCRIPTION"     },

{ COL_LABEL_CREATOR_ID,       "LABEL_CREATOR_ID"      },

{ COL_LABEL_CREATE_TS,        "LABEL_CREATE_TS"       },

{ COL_LABEL_PARENT_ID,        "LABEL_PARENT_ID"       },

{ COL_LABEL_VALUE,            "LABEL_VALUE"           },

{ COL_LABEL_VALUE_MODIFY_TS,  "LABEL_VALUE_MODIFY_TS" },

{ COL_DATA_LABEL_DATA_ID,     "DATA_LABEL_DATA_ID"    },

{ COL_DATA_LABEL_LABEL_ID,    "DATA_LABEL_LABEL_ID"   },

{ COL_DATA_LABEL_ASSIGN_TS,   "DATA_LABEL_ASSIGN_TS"  },

{ COL_COLL_LABEL_COLL_ID,     "COLL_LABEL_COLL_ID"    },

{ COL_COLL_LABEL_LABEL_ID,    "COLL_LABEL_LABEL_ID"   },

{ COL_COLL_LABEL_ASSIGN_TS,   "COLL_LABEL_ASSIGN_TS"  },
```

Code snippet 3: irods-src/server/genquery2/src/genquery2_sql.cpp code additions:

```
// Labels plugin tables (irods_labels_plugin)

"R_LABEL",                  // 20

"R_LABEL_VALUE",            // 21

"R_DATA_LABEL",             // 22

"R_COLL_LABEL",             // 23

// Labels plugin FK edges (irods_labels_plugin)

{to_index("R_LABEL"),      to_index("R_LABEL_VALUE")},

{to_index("R_LABEL"),      to_index("R_DATA_LABEL")},

{to_index("R_LABEL"),      to_index("R_COLL_LABEL")},

{to_index("R_DATA_LABEL"), to_index("R_DATA_MAIN")},

{to_index("R_COLL_LABEL"), to_index("R_COLL_MAIN")},

// Labels plugin join conditions (irods_labels_plugin)

{"{}.label_id = {}.label_id", 20},   // R_LABEL <-> R_LABEL_VALUE

{"{}.label_id = {}.label_id", 21},   // R_LABEL <-> R_DATA_LABEL

{"{}.label_id = {}.label_id", 22},   // R_LABEL <-> R_COLL_LABEL

{"{}.data_id = {}.data_id", 23},     // R_DATA_LABEL <-> R_DATA_MAIN

{"{}.coll_id = {}.coll_id", 24},     // R_COLL_LABEL <-> R_COLL_MAIN
```

ICOMMANDS TOOL: ILABEL

ilabel is an additional icommand, similar to the standard iRODS icommands [14] [15] — it is designed to be installed alongside iput, iget, imeta, and other built-in tools. It follows iRODS conventions: reads ~/.irods/irods_environment.json, uses the current iRODS session, and exits with standard return codes.

In every operation, it uses an RPC call to API #20100 on the server. The only client-side logic is resolving iRODS paths to object IDs via GenQuery before dispatching. This tool was developed to add client-side functionality to manage labels in a standard way within iRODS. The full list of commands includes add_label, get_label and others (fig. 4) (Code 4).

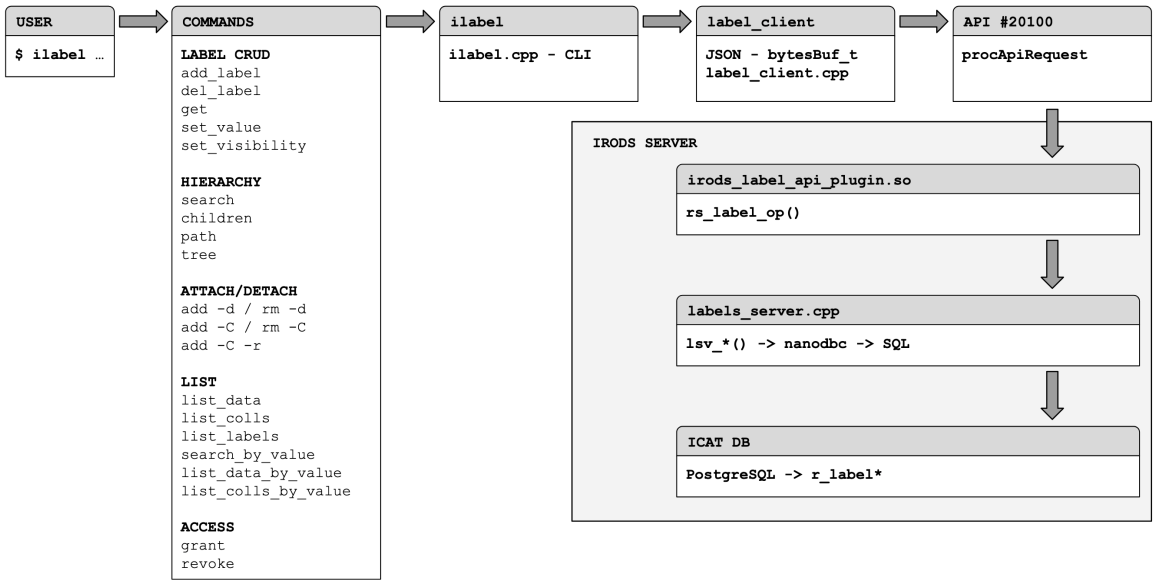

Figure 4: ilabel tool events flow diagram

Examples of the available commands are presented in Code 4.

Code snippet 4, ilabel commands example:

```
# Create

ilabel add_label project/2026 "Q1 experiment data"

ilabel add_label --parent project/2026 project/2026/raw "raw files"

ilabel add_label myLabel --visibility private

# Inspect

ilabel get       project/2026           # name, description, value, schema, timestamps

ilabel search    project/               # all labels matching prefix

ilabel children  project/2026           # direct children in the hierarchy

ilabel path      project/2026/raw       # full path from root

ilabel tree      project/               # entire subtree

# Mutate

ilabel set_visibility project/2026 private

ilabel set_value project/2026 '{"status":"active"}' --validation '{"type":"object"}'
```

```
ilabel del_label project/2026      # cascade-deletes objects and ACL rows

# What objects carry a given label?

ilabel list_data  project/2026          # data objects

ilabel list_colls project/2026          # collections

ilabel list_data_by_value  %active%     # data objects by value fragment

ilabel list_colls_by_value %active%     # collections by value fragment

# What labels are on a given object?

ilabel list_labels -d /zone/home/user/data.csv

ilabel list_labels -C /zone/home/user/experiments

ilabel search_by_value     %active%     # labels whose value matches fragment

# Access control

ilabel grant  project/2026 alice tempZone read

ilabel grant  project/2026 alice tempZone write

ilabel grant  project/2026 alice tempZone own

ilabel revoke project/2026 alice tempZone
```

PYTHON CLIENT IMPLEMENTATION

To demonstrate the compatibility of iRODS Labels with other iRODS clients, the implementation includes Python client scripts. The project implements two Python classes (fig. 6).

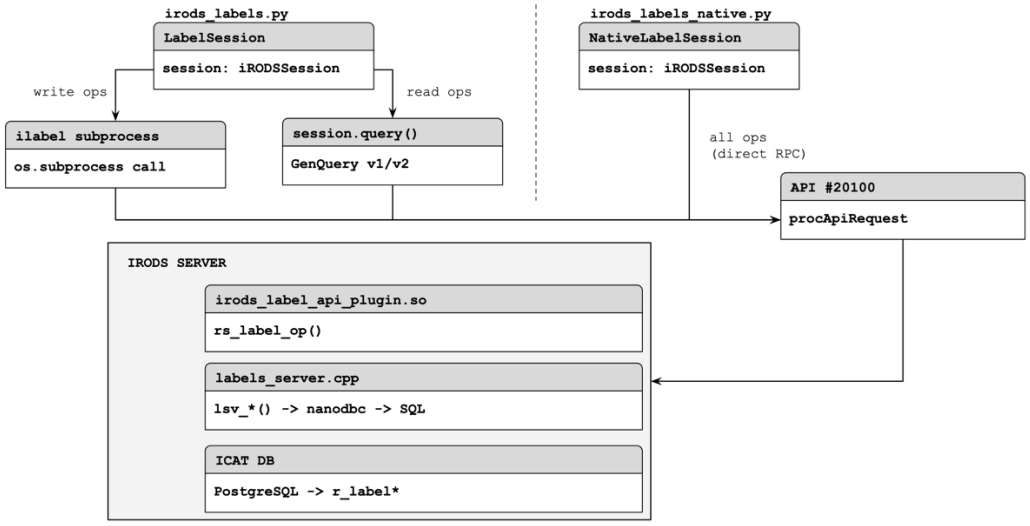

Figure 6: Python scripts events flow diagram for iRODS Labels implementation

The first class wraps ilabel command-line calls, and the second Python class uses the native RPC interface added to iRODS during the custom compilation.

The python-irodsclient has become one of the major clients used in iRODS-based software development, and this implementation for RPC calls demonstrates the usability of the concept. The code snippet 5 shows the usage of the RPC-based script:

Code snippet 5: Python code snippet of native RPC usage for iRODS Labels

```
from irods_labels_native import NativeLabelSession

with NativeLabelSession(session) as ls:

    ls.add_label("project/2026","Experiment data for Q1")

    ls.attach_to_data("project/2026", "/zone/home/user/data.csv")

    ls.grant("project/2026", "alice", "tempZone", "read")

    results = ls.list_data_by_label("project/2026")
```

CONCLUSION

iRODS Labels is an addition to, not a replacement for, the existing, well-proven and widely used AVUs functionality. It solves a number of practical use cases in dealing with metadata, especially in the context of using metadata templates that usually require template structure validation and data verification. There are certain challenges with integrating this functionality directly into iRODS at the API level, but the proposed solution shows that the actual changes at the core iRODS level are minimal. The principal results of the presented project are:

- A new plugin that compiles successfully within the iRODS core
- Labels provide additional functionality for tagging and data management within iRODS
- iRODS Labels is not a replacement but rather an addition to the existing AVUs mechanism, which has its own value and purpose

The next development steps include:

- Adding MySQL and Oracle databases support
- db_plugin hook verification for locks, or FK integration as an alternative
- Cross-tree references — one label with multiple parent labels
- Label value versioning — to preserve the history of template changes

Ultimately, the goal is to propose iRODS Labels to the iRODS Consortium for integration into the iRODS core, which would eliminate the need for a custom compilation process and make the functionality available to all iRODS users out of the box.

ACKNOWLEDGMENTS

The author would like to thank the University of Groningen and the RDMS Team colleagues for their support during the project and the preparation of the paper.

REFERENCES

[1] M. D. Wilkinson et al., "The FAIR Guiding Principles for scientific data management and stewardship," Sci. Data, vol. 3, Art. no. 160018, Mar. 2016, doi: 10.1038/sdata.2016.18.

[2] European Parliament and Council of the European Union, "Regulation (EU) 2016/679 (General Data Protection Regulation)," *Official Journal of the European Union*, vol. L119, pp. 1–88, May 2016. **https://eur-lex.europa.eu/eli/reg/2016/679/oj/eng**

[3] P. Voigt and A. von dem Bussche, *The EU General Data Protection Regulation (GDPR): A Practical Guide*. Cham, Switzerland: Springer, 2017. doi: 10.1007/978-3-319-57959-7.

[4] McMahon, B. (2024). The evolution of raw data archiving and the growth of its application to scientific peer review. Acta Crystallographica Section A: Foundations and Advances, 80(4), 211–223. doi: 10.1107/S205225252400455X

[5] iRODS Consortium, "Control your data: iRODS, the integrated Rule-Oriented Data System," RENCI, University of North Carolina at Chapel Hill, NC, USA, White Paper, Jul. 2014. [Online]. Available: **https://irods.org/uploads/2014/07/iRODS-White-Paper-2014.pdf**

[6] iRODS, **https://irods.org/**

[7] iRODS documentation. **https://docs.irods.org/**

[8] iRODS Consortium. (2017). *Spec White Paper: Metadata Templates*. iRODS. **https://irods.org/uploads/2017/20170315-MetadataTemplates-Whitepaper.pdf**

[9] M. Chua, A. de Torcy, J. H. Ward, and J. Crabtree, "Using iRODS to preserve and publish a Dataverse archive," *Data Intensive Cyber Environments Center*, Tech. Rep., 2010. Available: **https://irods.org/uploads/2010/Chua-Dataverse_Archive-paper.pdf**

[10] M. Montes and P. Borgermans, "Towards rich and standardized metadata in iRODS," in Proc. iRODS User Group Meeting, Chapel Hill, NC, USA, 2023, pp. 1–10. Available: **https://irods.org/uploads/2023/Montes-KULeuven-Towards_rich_and_standardized_metadata_in_iRODS-paper.pdf**

[11] A. Tsyganov, S. Stoica, M. Babai, V. Soancatl-Aguilar, and J. Mc Farland, "The Research Data Management System at the University of Groningen (RUG RDMS): Architecture, solution engines and challenges," in *Proc. iRODS User Group Meeting*, 2021, pp. 45–58. **https://irods.org/uploads/2021/Tsyganov-Stoica-Strikwerda-UnivGroningen-Research_Data_Management_System_at_Univ_of_Groningen_Architecture_Solution_Engines_and_Challenges-paper.pdf**

[12] The University of Groningen, **https://www.rug.nl/**

[13] Anthropic, "Introducing Claude Sonnet 4.6," Anthropic News, Feb. 17, 2026. Available: **https://www.anthropic.com/news/claude-sonnet-4-6**

[14] A. Rajasekar *et al.*, *iRODS Primer: Integrated Rule-Oriented Data System*, 2nd ed. Springer Nature, 2022. doi: 10.1007/978-3-031-02271-5.

[15] A. Rajasekar, M. Wan, R. Moore, and W. Schroeder, "integrated Rule-Oriented Data System Reference Manual," San Diego Supercomputer Center (SDSC), Tech. Rep., 2009. Available: **https://irods.org/uploads/2009/iRODS-refmanual.pdf**