Digital Libraries, Preservation Environments, and Data Grids

Reagan W. Moore
San Diego Supercomputer Center

moore@sdsc.edu
http://irods.sdsc.edu
http://www.sdsc.edu/srb/
Opportunity

- Digital libraries typically assume a local storage environment
- Preservation environments typically assume a centralized storage resource
- Data grid technology provides the infrastructure that organizes distributed data into shared collections, supports replication, and supports federation
Current Integration Activity

• DSpace
  • Digital library services for ingestion, curation, organization of records
  • Port on top of SRB & iRODS data grids
• Enables new capabilities for DSpace
  • Store larger collections than size of local disk
  • Replicate data for integrity assurance
  • Federate multiple DSpace systems (access data within another DSpace digital library)
Next Generation Data Grid

- User asks for data
- Data request goes to iRODS Server
- Server looks up information in catalog
- Catalog tells which iRODS server has data
- 1st server asks 2nd for data
- The 2nd iRODS server applies rules
Current Integration Activity

• Fedora digital library middleware
  • Manage relationships on digital entities
  • Associate behaviors with digital entities
  • Port on top of iRODS data grid

• Enables new capabilities for Fedora
  • Enforcement of management policies
  • Validation of integrity assertions
  • Automation of administrative tasks
# Future of Data Management

**iRODS - integrated Rule-Oriented Data System**

<table>
<thead>
<tr>
<th>Data Management Environment</th>
<th>Conserved Properties</th>
<th>Control Mechanisms</th>
<th>Remote Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management Functions</td>
<td>Assessment Criteria</td>
<td>Management Policies</td>
<td>Management Processes</td>
</tr>
</tbody>
</table>

**Data grid's Management virtualization**

<table>
<thead>
<tr>
<th>Data Management Infrastructure</th>
<th>Persistent State</th>
<th>Rules</th>
<th>Micro-services</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data grid's Data and trust virtualization</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Physical Infrastructure</th>
<th>Database</th>
<th>Rule Engine</th>
<th>Storage System</th>
</tr>
</thead>
</table>

---

**Infrastructure**

- Database
- Rule Engine
- Storage System
Why is this important?

• Organizing scientific data into collections
  • Sizes measured in the tens of millions of files
• Associating descriptive metadata with scientific data
  • Namelist input parameters
  • Processing steps
• Assigning required access controls
  • Human subject approval flags for IRB
  • HIPAA patient confidentiality
  • Time dependent access controls
<table>
<thead>
<tr>
<th>Project</th>
<th>5/17/02 GBs of data stored</th>
<th>1000Os of files</th>
<th>6/30/04 GBs of data stored</th>
<th>1000Os of files</th>
<th>Users with ACLs</th>
<th>11/29/07 GBs of data stored</th>
<th>1000Os of files</th>
<th>Users with ACLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSF / NVO</td>
<td>17,800</td>
<td>5,139</td>
<td>51,380</td>
<td>8,690</td>
<td>80</td>
<td>88,216</td>
<td>14,550</td>
<td>100</td>
</tr>
<tr>
<td>NSF / NPACI</td>
<td>1,972</td>
<td>1,083</td>
<td>17,578</td>
<td>4,694</td>
<td>380</td>
<td>39,697</td>
<td>7,590</td>
<td>380</td>
</tr>
<tr>
<td>Hayden</td>
<td>6,800</td>
<td>41</td>
<td>7,201</td>
<td>113</td>
<td>178</td>
<td>8,013</td>
<td>161</td>
<td>227</td>
</tr>
<tr>
<td>Pzone</td>
<td>438</td>
<td>31</td>
<td>812</td>
<td>47</td>
<td>49</td>
<td>28,799</td>
<td>17,640</td>
<td>68</td>
</tr>
<tr>
<td>NSF / LDAS-SALK</td>
<td>239</td>
<td>1</td>
<td>4,562</td>
<td>16</td>
<td>66</td>
<td>207,018</td>
<td>169</td>
<td>67</td>
</tr>
<tr>
<td>NSF / SLAC-JCSG</td>
<td>514</td>
<td>77</td>
<td>4,317</td>
<td>563</td>
<td>47</td>
<td>23,854</td>
<td>2,493</td>
<td>55</td>
</tr>
<tr>
<td>NSF / TeraGrid</td>
<td></td>
<td></td>
<td>80,354</td>
<td>685</td>
<td>2,962</td>
<td>282,536</td>
<td>7,257</td>
<td>3,267</td>
</tr>
<tr>
<td>NIH / BIRN</td>
<td>5,416</td>
<td>3,366</td>
<td>148</td>
<td></td>
<td></td>
<td>20,400</td>
<td>40,747</td>
<td>445</td>
</tr>
<tr>
<td>NCAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>70,334</td>
<td>325</td>
<td>2</td>
</tr>
<tr>
<td>LCA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3,787</td>
<td>77</td>
<td>2</td>
</tr>
<tr>
<td>Digital Library</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSF / LTER</td>
<td>158</td>
<td>3</td>
<td>233</td>
<td>6</td>
<td>35</td>
<td>260</td>
<td>42</td>
<td>36</td>
</tr>
<tr>
<td>NSF / Portal</td>
<td>33</td>
<td>5</td>
<td>1,745</td>
<td>48</td>
<td>384</td>
<td>2,620</td>
<td>53</td>
<td>460</td>
</tr>
<tr>
<td>NIH / AfCS</td>
<td>27</td>
<td>4</td>
<td>462</td>
<td>49</td>
<td>21</td>
<td>733</td>
<td>94</td>
<td>21</td>
</tr>
<tr>
<td>NSF / SIO Explorer</td>
<td>19</td>
<td>1</td>
<td>1,734</td>
<td>601</td>
<td>27</td>
<td>2,750</td>
<td>1,202</td>
<td>27</td>
</tr>
<tr>
<td>NSF / SCEC</td>
<td></td>
<td></td>
<td>15,246</td>
<td>1,737</td>
<td>52</td>
<td>168,931</td>
<td>3,545</td>
<td>73</td>
</tr>
<tr>
<td>LLNL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18,934</td>
<td>2,338</td>
<td>5</td>
</tr>
<tr>
<td>CHRON</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12,863</td>
<td>6,443</td>
<td>5</td>
</tr>
<tr>
<td>Persistent Archive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NARA</td>
<td>7</td>
<td>2</td>
<td>63</td>
<td>81</td>
<td>58</td>
<td>5,023</td>
<td>6,430</td>
<td>58</td>
</tr>
<tr>
<td>NSF / NSDL</td>
<td></td>
<td></td>
<td>2,785</td>
<td>20,054</td>
<td>119</td>
<td>7,499</td>
<td>84,984</td>
<td>136</td>
</tr>
<tr>
<td>UCSD Libraries</td>
<td></td>
<td></td>
<td>127</td>
<td>202</td>
<td>29</td>
<td>5,205</td>
<td>1,328</td>
<td>29</td>
</tr>
<tr>
<td>NHPRC / PAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2,576</td>
<td>966</td>
<td>28</td>
</tr>
<tr>
<td>RoadNet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3,557</td>
<td>1,569</td>
<td>30</td>
</tr>
<tr>
<td>UCTV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7,140</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>LOC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6,644</td>
<td>192</td>
<td>8</td>
</tr>
<tr>
<td>Earth Sci</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6,136</td>
<td>652</td>
<td>5</td>
</tr>
<tr>
<td>TOTAL</td>
<td>28 TB</td>
<td>6 mil</td>
<td>194 TB</td>
<td>40 mil</td>
<td>4,635</td>
<td>1,023 TB</td>
<td>200 mil</td>
<td>5,539</td>
</tr>
</tbody>
</table>
iRODS Wiki

- [http://irods.sdsc.edu](http://irods.sdsc.edu)
- Descriptions of the technology
- Publications / presentations
- Download
- Performance tests
- Tinderbox system (tracks upgrades)
- irods-chat page
For More Information

Reagan W. Moore
San Diego Supercomputer Center
moore@sdsc.edu

http://www.sdsc.edu/srb/
http://irods.sdsc.edu/
iRODS
Class Exercise

- [http://irods.sdsc.edu](http://irods.sdsc.edu)
  - Downloads
    - BSD license
    - Registration / agreement
  - Tar file
    - Installation script (Linux, Solaris, Mac OS X)
    - Automated download of PostgreSQL, ODBC
    - Installation of PostgreSQL, ODBC, iRODS
    - Initiation of iRODS collection
• **Unpack** the release tar file
  • `gzip -d irods.tar`
  • `tar xf irods.tar`

• **cd** into the top directory and execute
  • `./irodssetup`

• **It will prompt for a few parameters**
Set up iRODS

iRODS is a flexible data archive management system that supports many different site configurations. This script will ask you a few questions, then automatically build and configure iRODS.

There are four main components to iRODS:

1. An iRODS server that manages stored data.
2. An iCAT catalog that manages metadata about the data.
3. A database used by the catalog.
4. A set of 'i-commands' for command-line access to your data.

You can build some, or all of these, in a few standard configurations. For new users, we recommend that you build everything.
Input Parameters

- iRODS configuration:
  - Build an iRODS server? (yes/no) yes
  - Include an iCAT catalog? (yes/no) yes

- For security reasons, the build process will create a new iRODS administrator account named 'rods' for managing the system.

- Enter a new password for the iRODS account? (password) xyyyyyy
Input Parameters

- Database configuration:
- -----------------------
- The iCAT uses a database to store metadata. You can build and configure a new Postgres database now or use an existing database.
- **Build Postgres?** (yes/no) yes
- You can select the directory for Postgres:
  - If you are creating a new iRODS installation, select a new directory. Postgres will be automatically downloaded, built, and installed there.
  - If you are upgrading an iRODS installation and wish to re-use an existing database, enter the path to that Postgres directory.
- **Where should Postgres be installed?** (directory path)
  /Applications/Postgres
- For security reasons, the new database will create an administrator account for 'reaganmoore' and assign a password.
- **Enter a password for the new database account?** (password)
  xxxxxxxxxx
The iRODS build and setup is ready to begin.

- iRODS server: build
- account 'rods'
- password 'taipei'
- path '/Applications/iRODS'

- iCAT catalog: build

- Postgres: install a new database
- enable iRODS scripts to start/stop database
- account 'reaganmoore'
- password 'taipei'
- path '/applications/irods/postgresql'

- I-commands: build

- Ready? (yes/no) yes
Installation

Class Exercise

- Track the completion status of each step:
- Preparing...
- Installing Postgres database...
  - Step 1 of 4: Preparing to install...
  - Step 2 of 4: Installing Postgres... About 11 minutes
  - Step 3 of 4: Installing UNIX ODBC... About 26 minutes
  - Step 4 of 4: Setting up Postgres...
  - Step 5 of 4: Setting up iRODS...
- Configuring iRODS... About 1 minute
  - Step 1 of 5: Enabling modules...
  - Step 2 of 5: Verifying configuration...
  - Step 3 of 5: Checking host system...
  - Step 4 of 5: Updating configuration files...
  - Step 5 of 5: Cleaning out previously compiled files...
- Compiling iRODS... About 3 minutes
  - Step 1 of 3: Compiling library and i-commands...
  - Step 2 of 3: Compiling iRODS server...
  - Step 3 of 3: Compiling tests...
Using a Data Grid – *in Abstract*

- User asks for data from the data grid
- The data is found and returned
- Where & how details are hidden
Data Grid Capabilities

• Logical file name space
  • Directory hierarchy / soft links
  • Versions / backups / replicas
  • Aggregation / containers
  • Descriptive metadata
  • Digital entities

• Authentication and authorization
  • GSI, challenge-response, Shibboleth
  • ACLs, audit trails
  • Checksums, synchronization
  • Logical user name space
  • Aggregation / groups
Data Grid Capabilities

• **Logical storage names**
  • Dynamic resource creation
  • Standard operations
  • Heterogeneous storage systems
  • Trash
  • Collective operations / storage groups

• **Data transport**
  • Parallel I/O
  • Small file transport
  • Message engine
  • Containers / tar files / HDF5
  • Aggregation of I/O commands - remote procedures
Data Grid Capabilities

- Remote procedures
  - Atomic / deferred / periodic
  - Procedure execution / chaining
  - Structured information
- Structured information
  - Metadata catalog interactions / 205 queries
  - Information transmission
  - Template parsing
  - Memory structures
  - Report generation / audit trail parsing
Data Grid Capabilities

• Rules
  • User / administrative / internal
  • Remote web service invocation
  • Rule & micro-service creation
  • Standards / XAM, SNIA

• Installation
  • CVS / modules
  • System dependencies
  • Automation
Data Grids

• Administration
  • User creation
  • Resource creation
  • Token management
  • Listing

• Collaborations
  • Development plans
  • International collaborators
Generalizations

- Logical names
- Aggregation on each name space
- Virtualization levels
- Automation
iRODS Tutorials - 2008

- January 31, SDSC
- April 8 - ISGC, Taipei
- May 13 - China, National Academy of Science
- May 27-30 - UK eScience, Edinburgh
- June 5 - OGF23, Barcelona
- July 7-11 - SAA, SDSC
- August 4-8 - SAA, SDSC
- August 25 - SAA, San Francisco
Tutorial Topics

- Data management applications
  - Concepts behind distributed data management
- Data grids
  - Infrastructure independence
- Rule-based data grids
  - Automation of management policies
- Open source software
  - iRODS - integrated Rule-Oriented Data System
  - Installation and use of iRODS data grid
Data Management Applications
(What do they have in common?)

• Data grids
  • **Share data** - organize distributed data as a collection
• Digital libraries
  • **Publish data** - support browsing and discovery
• Persistent archives
  • **Preserve data** - manage technology evolution
• Real-time sensor systems
  • **Federate sensor data** - integrate across sensor streams
• Workflow systems
  • **Analyze data** - integrate client- & server-side workflows
Approach

• To meet the diverse requirements, the architecture must:
  • Be highly modular
  • Be highly extensible
  • Provide infrastructure independence
  • Enforce management policies
  • Provide scalability mechanisms
  • Manipulate structured information
  • Enable community standards
Generic Infrastructure

• Data grids manage data distributed across multiple types of storage systems
  • File systems, tape archives, object ring buffers

• Data grids manage collection attributes
  • Provenance, descriptive, system metadata

• Data grids manage technology evolution
  • At the point in time when new technology is available, both the old and new systems can be integrated
Extremely Successful

- Storage Resource Broker (SRB) manages 2 PBs of data in internationally shared collections
- Data collections for NSF, NARA, NASA, DOE, DOD, NIH, LC, NHPRC, IMLS: APAC, UK e-Science, IN2P3, WUNgrid
  - Astronomy Data grid
  - Bio-informatics Digital library
  - Earth Sciences Data grid
  - Ecology Collection
  - Education Persistent archive
  - Engineering Digital library
  - Environmental science Data grid
  - High energy physics Data grid
  - Humanities Data Grid
  - Medical community Digital library
  - Oceanography Real time sensor data, persistent archive
  - Seismology Digital library, real-time sensor data
- Goal has been generic infrastructure for distributed data
iRODS vs Storage Resource Manager

**SRB - manage data**
- Quotas
  - Data redirection to alternative
- Logical name
  - Transfer on logical name
- Synchronous/Asynchronous
  - Deferred operations
- Clients
  - GridFTP, SRB, Web, Library
- Compound resource
  - Stage
  - Cache (automated)

**SRM - manage storage**
- Reservation
  - Fail if not enough space
- SURL
  - Transfer on TURL
- Asynchronous
  - Pole for status
- Clients
  - GridFTP
- Compound resource
  - Stage
Tension between Common and Unique Components

- **Synergism - common infrastructure**
  - Distributed data
    - Sources, users, performance, reliability, analysis
  - Technology management
    - Incorporate new technology

- **Unique components - extensibility**
  - Information management
    - Semantics, formats, services
  - Management policies
    - Integrity, authenticity, availability, authorization
Observations of Production Data Grids

• Each community implements different management polices
  • Community specific preservation objectives
  • Community specific assertions about properties of the shared collection
  • Community specific management policies

• Need a mechanism to support the socialization of shared collections
  • Map from assertions made by collection creators to expectations of the users
Data Grid Mechanisms

• Implement essential components needed for synergism
  • Storage Resource Broker - SRB
  • Infrastructure independence
  • Data and trust virtualization

• Implement components needed for specific management policies and processes
  • integrated Rule Oriented Data System - iRODS
  • Policy management virtualization
  • Map processes to standard micro-services
  • Structured information management and transmission
Data Management

• Observe that efficient systems require
  • Aggregation of data
    • Bulk operations
    • Bulk storage
  • Aggregation of operations on data
    • Management policies
    • Remote Procedures
    • Structured information
    • Posix I/O

• iRODS maps from policies to procedures to operations on structured information to I/O operations on storage systems
iRODS Source Distribution

- INSTALL.txt
- LICENSE.txt
- Makefile
- README.txt
- Configure
- Vault

- irodsctl
- irodsssetup

- COPYRIGHT
- CVS
- bin
- clients
- config
- doc
- install
- installLogs
- lib
- modules
- nt
- scripts
- server
User Configuration

• To use the iRODS 'i-commands', update your PATH:
  • For csh users:
    • set path=('/Applications/iRODS/clients/icommands/bin $path')
  • For sh or bash users:
    • PATH=/Applications/iRODS/clients/icommands/bin:$PATH
• To start and stop the servers, use 'irodsctl':
  • irodsctl start
  • irodsctl stop
  • irodsctl restart
• Add '--help' for a list of commands.
irodsctl options

• Usage is:
  • /Applications/iRODS/scripts/perl/irodsctl.pl [options] [commands]

• Help options:
  • --help Show this help information

• Verbosity options:
  • --quiet Suppress all messages
  • --verbose Output all messages (default)

• iRODS server Commands:
  • istart Start the iRODS servers
  • istop Stop the iRODS servers
  • irestart Restart the iRODS servers
**irodsctl options**

- **Database commands:**
  - `dbstart` Start the database servers
  - `dbstop` Stop the database servers
  - `dbrestart` Restart the database servers
  - `dbdrop` Delete the iRODS tables in the database
  - `dboptimize` Optimize the iRODS tables in the database
  - `dbvacuum` Same as 'optimize'

- **General Commands:**
  - `start` Start the iRODS and database servers
  - `stop` Stop the iRODS and database servers
  - `restart` Restart the iRODS and database servers
  - `status` Show the status of iRODS and database servers
  - `test` Test the iRODS installation
Environment Variables

• In home directory
  • cd ~.irods
  • vi .irodsEnv
# iRODS personal configuration file.

# This file was automatically created during iRODS installation.
#   Created Fri Jan 18 10:01:48 2008
#

# iRODS server host name:
irodsHost '140.109.127.17'
# iRODS server port number:
irodsPort 1247
# Default storage resource name:
irodsDefResource 'demoResc'
# Home directory in iRODS:
irodsHome '/tempZone/home/user1'
# Current directory in iRODS:
irodsCwd '/tempZone/home/user1'
# Account name:
irodsUserName 'user1'
# Zone:
irodsZone 'tempZone'
Directory ~irods/server

Class Exercise

- `ls -l`
- `total 32`
- `drwxr-sr-x 5 asdasd admin 170 Oct 3 16:10 CVS`
- `-rw-r--r-- 1 asdasd admin 8906 Sep 28 16:52 Makefile`
- `-rw-r--r-- 1 asdasd admin 281 Sep 12 15:28 README.txt`
- `drwxr-sr-x 7 asdasd admin 238 Oct 3 16:10 api`
- `drwxr-sr-x 11 asdasd admin 374 Oct 15 16:34 bin`
- `drwxr-sr-x 12 asdasd admin 408 Oct 15 16:35 config`
- `drwxr-sr-x 7 asdasd admin 238 Oct 3 16:10 core`
- `drwxr-sr-x 7 asdasd admin 238 Oct 3 16:10 drivers`
- `drwxr-sr-x 7 asdasd admin 238 Oct 3 16:10 icat`
- `drwxr-sr-x 5 asdasd admin 170 Oct 15 16:35 log`
- `drwxr-sr-x 7 asdasd admin 238 Oct 3 16:10 re`
- `drwxr-sr-x 4 asdasd admin 136 Oct 3 16:10 rules`
- `drwxr-sr-x 4 asdasd admin 136 Oct 3 16:10 schema`
- `drwxr-sr-x 8 asdasd admin 272 Oct 3 16:10 test`
Directory ~irods/server/bin

Class Exercise

• $ ls -l server/bin
• total 28176
• drwxr-xr-x 5 reaganmo admin 170 Jan 18 08:39 CVS
• drwxr-xr-x 5 reaganmo admin 170 Jan 18 08:39 cmd
• -rwxr-xr-x 1 reaganmo admin 3604048 Jan 18 10:01 irodsAgent
• -rwxr-xr-x 1 reaganmo admin 3598516 Jan 18 10:01 irodsReServer
• -rwxr-xr-x 1 reaganmo admin 3611264 Jan 18 10:01 irodsServer
• -rwxr-xr-x 1 reaganmo admin 3598024 Jan 18 10:01 irodsXmsgServer
• -rwxr-xr-x 1 reaganmo admin 1655 Sep 12 15:28 list.pl
• -rwxr-xr-x 1 reaganmo admin 3400 Sep 12 15:28 vacuumdb.pl
Directory ~irods/server/config

Class Exercise

- $ ls -l server/config
- total 48
- drwxr-sr-x 5 asdasd admin 170 Oct 3 16:10 CVS
- -rw-r--r-- 1 asdasd admin 782 Sep 12 15:28 HostAccessControl
- -rw-r--r-- 1 asdasd admin 162 Sep 12 15:28 README.txt
- -rw-r--r-- 1 asdasd admin 665 Sep 12 15:28 irodsHost
- -rw-r--r-- 1 asdasd admin 665 Sep 12 15:28 irodsHost.in
- drwxr-sr-x 3 asdasd admin 102 Oct 3 16:10 packedRei
- drwxr-sr-x 22 asdasd admin 748 Oct 3 16:10 reConfigs
- -rw------- 1 asdasd admin 951 Oct 15 16:35 server.config
- -rw-r--r-- 1 asdasd admin 970 Sep 12 15:28 server.config.in
- -rw-r--r-- 1 asdasd admin 0 Oct 15 16:32 server.config.sav
Directory `~irods/server/config/reconfigs`

- `$ ls -l server/config/reconfigs`
- total 216
- `drwxr-sr-x  5 asdasd admin  170 Oct  3 16:10 CVS`
- `-rw-r--r--  1 asdasd admin  4102 Sep 12 15:28 core.dvm`
- `-rw-r--r--  1 asdasd admin   763 Sep 19 15:19 core.fn.m`
- `-rw-r--r--  1 asdasd admin  14384 Oct  3 12:32 core.irb`
- `-rwxr-xr-x  1 asdasd admin   192 Sep 12 15:28 core.irb.1`
- `-rw-r--r--  1 asdasd admin  227 Sep 12 15:28 core.irb.2`
- `-rw-r--r--  1 asdasd admin  101 Sep 12 15:28 core.irb.3`
- `-rwxr-xr-x  1 asdasd admin  14157 Sep 19 15:34 core.irb.orig`
- `-rw-r--r--  1 asdasd admin  4102 Oct  3 12:32 core2.dvm`
- `-rw-r--r--  1 asdasd admin  763 Oct  3 12:32 core2.fn.m`
- `-rw-r--r--  1 asdasd admin  690 Sep 12 15:28 core2.irb`
- `-rw-r--r--  1 asdasd admin  714 Sep 12 15:28 core3.irb`
- `-rw-r--r--  1 asdasd admin  777 Sep 26 10:08 core4.irb`
- `-rw-r--r--  1 asdasd admin  269 Sep 12 15:28 misc.irb`
- `-rw-r--r--  1 asdasd admin  1275 Sep 12 15:28 nara.irb`
- `-rw-r--r--  1 asdasd admin  745 Sep 12 15:28 nvo.irb`
- `-rw-r--r--  1 asdasd admin  619 Sep 12 15:28 raja.irb`
- `-rw-r--r--  1 asdasd admin  750 Sep 12 15:28 raja2.irb`
- `-rw-r--r--  1 asdasd admin  2315 Sep 12 15:28 rajatest.irb`
- `-rw-r--r--  1 asdasd admin  1372 Sep 12 15:28 reRules`
iRODS Clients

- **Currently seven clients**
  - iRODS rich web client
  - Unix shell commands
    - iRODS/clients/icommands/bin
  - FUSE user level file system
    - iRODS/clients/fuse/bin/irodsFs fmount
  - Jargon Java I/O class library
    - iRODS/java/jargon
  - PHP web browser and PHP client library
    - [http://irods.sdsc.edu](http://irods.sdsc.edu)
  - C library calls
  - Parrot user level file system
    - Douglas Thain, Notre Dame University
Class Exercise

- `ls -l clients/icommands/bin`
- `total 28296`
- `total 28296`
- `total 28296`
- `total 28296`
iCommands
~/irods/clients/icommands/bin

- icd
- ichmod
- icp
- ils
- imkdir
- imv
- ipwd
- irm
- ienv
- ierrror

- iget
- iput
- ireg
- irepl
- itrim
- irsync
- ilsresc
- iphymv
- irmtrash
- ichksum
- iinit
- iexit

- iqdel
- iqmod
- iqstat
- iexecmd
- irule
- iuserinfo
- isysmeta
- imeta
- iquest
- imiscsvrinfo
- iadmin
iRODS Components

- Clients
- Persistent state information catalog - iCAT
- Server middleware at each storage system
- Rule engine at each storage system

- Implements server-side workflows composed from micro-services
- Rules control execution of micro-services
iRODS Extensibility

• Rules
  • Use default rules for data grid capabilities
  • Administrator modification of pre-packaged rules (turn capabilities on and off)
  • Creation of new rules using existing micro-services
  • Write new micro-services and the rules controlling their execution
iRODS Extensibility

• State information
  • Use existing system state information, audit trails
  • Add user-defined metadata (descriptive context)
  • Create schema versions (map persistent state name to a different column in the database)
  • Add new system metadata
iRODS Extensibility

• **Drivers**
  - Add drivers for new storage protocols
  - Mounted Collection interface, add drivers to interact with other data management systems to retrieve information required for operations

• **APIs**
  - Add new client types on top of C-library, Unix i-commands, and Java class library

• **Functionality**
  - Add micro-services
  - Extend Posix I/O by adding functions to framework
Connecting to iRODS Collection

Class Exercise

- `iinit` - initiate connection using default parameters specified in the file `~/.irods/.irodsEnv`
  - `irodsHost` `srbbrick14.sdsc.edu`
  - `irodsPort` 1247
  - `irodsDefResource` `demoResc`
  - `irodsHome` `/tempZone/home/rods`
  - `irodsCwd` `/tempZone/home/rods`
  - `irodsUserName` 'rods'
  - `irodsZone` 'tempZone'

- `ienv` - lists the contents of the `.irodsEnv` file

- Authentication done using the file `~/.irods/.irodsA`
  - Created when you do an `iinit`
Connect to iRODS

- $ iinit -h
- Creates a file containing your iRODS password in a scrambled form, to be used automatically by the icommands.

Usage: iinit [-ehvVl]
  - -e echo the password as you enter it (normally there is no echo)
  - -l list the iRODS environment variables (only)
  - -v verbose
  - -V Very verbose
  - -h this help
Disconnect From iRODS

Class Exercise

- $ iexit -h
- Exits iRODS session (cwd) and optionally removes the scrambled password file produced by iinit.
- Usage: iexit [-vh] [full]
- If 'full' is included the scrambled password is also removed.
- -v verbose
- -V very verbose
- -h this help
Data Grids

- **Data virtualization**
  - Provide the persistent, global identifiers needed to manage distributed data
  - Provide standard operations for interacting with heterogeneous storage system
  - Map from storage protocols to preferred clients

- **Trust virtualization**
  - Manage authentication and authorization
  - Enable access controls on data, metadata, storage

- **Federation**
  - Controlled sharing of name spaces, files, and metadata between independent data grids
  - Data grid chaining / Central archives / Master-slave data grids / Peer-to-Peer data grids
Data Virtualization

Data Access Methods (C library, Unix, Web Browser)

Data Collection

Storage Repository
- Storage location
- User name
- File name
- File context (creation date, …)
- Access controls

Data Grid
- Logical resource name space
- Logical user name space
- Logical file name space
- Logical context (metadata)
- Access constraints

Data is organized as a shared collection
$ ils -l
/temp Zone/home/user1:

$ imkdir nvo
$ imkdir tg
$ imkdir looptest
$ ils -l

Do you see the new directories?
$ ils -l

tempZone/home/user1:
  - /tempZone/home/user1/loopTest
  - /tempZone/home/user1/nvo
  - /tempZone/home/user1/tg

$ iput ../src/icp.c nvo/icp.c
Listing File Information

$ ils -l nvo
/tempZone/home/user1/nvo:
    user1 0 demoResc 3693 2008-01-22 16:59 & icp.c

$ ils -L nvo
/tempZone/home/user1/nvo:
    user1 0 demoResc 3693 2008-01-22 16:59 & icp.c
/Applications/iRODS/Vault/user1/nvo/icp.c

../../server/bin/stop.pl
../../server/bin/start.pl
iRODS Web Interface

• http://irods.sdsc.edu
• Click on “web client” in left hand box
  • Two web clients are listed
    • Web Client - Python-based browser
    • iRODS Web Client - PHP-Ajax based browser
Python-based Web Client

• Basic interface for multi-browser support
  • http://pho27.sdsc.edu/rodsclient/
iROD Web Client

• Click on iRODS web Client
  • Then on bottom of page, click on
  • https://rt.sdsc.edu:8443/irods/index.php
Password is taipei
iadmin - Main iRODS Administrator Interface

- Interactive or command-line interface
  - A blank execute line invokes the interactive mode, where it prompts and executes commands until 'quit' or 'q' is entered. Single or double quotes can be used to enter items with blanks.

- Manages
  - users, user-groups, passwords, resources, resource-groups, directories, database, tokens
iadmin
Class Exercise

• iadmin -h - Command line
• iadmin - Interactive mode
• h - help, list commands
• q - quit
iadmin - Main subcommands

- lu - list user
- lr - list resource
- ls - list files
- lz - list zone
- lg - list group
- lgd - list group details
- lrg - list resource group
- lt - list token
- lf - list file details
- mkuser - make user
- moduser - modify user
- rmuser - remove user
- mkresc - make resource
- modresc - modify resource
- rmresc - remove resource
- mkgroup - make group
- rmgroup - remove group
- atg - add to group
- rfg - remove from group
- atrg - add (resource) to resource group
- rfrg - remove (resource) from resource group
- at - add token
- rt - remove token
- pv - run a periodic vacuum
Class Exercise

- iadmin - Use interactive mode
- mkuser u2 badtype - Create new user & user type
- lt - List tokens for allowed type
- lt user_type - List allowed user types
- mkuser u2 rodsuser - Create the user
- lu - List users
- lu u2 - List user u2
- moduser u2 password [pass] - set password
- q - quit
iRODS Storage Name Space

Class Exercise

$ ilsresc -l

resource name: demoResc
resc id: 10007
zone: tempZone
type: unix file system
class: archive
location: 140.109.127.17
vault: /Applications/iRODS/Vault
free space:
info:
comment:
create time: 01207540416: 2008-04-06.20:53:36
modify time: 01207540416: 2008-04-06.20:53:36
$ iuserinfo
name: user1
id: 10009
type: rodsuser
zone: tempZone
dn:
info:
comment:
create time: 01207550000: 2008-04-06 23:33:20
member of group: user1
Data Virtualization

- **Access Interface**
- **Standard Micro-services**
- **Data Grid**
- **Standard Operations**
- **Storage Protocol**
- **Storage System**

Map from the actions requested by the access method to a standard set of micro-services. The standard micro-services are mapped to the operations supported by the storage system.
Standard Operations

• The capabilities needed to interact with storage systems
  • Posix I/O
  • File manipulation
  • Metadata manipulation
  • Bulk operations
  • Parallel I/O
  • Remote procedures
  • Registration
$ iput -h

**Usage:** iput [-fkKrvV] [-D dataType] [-N numThreads] [-n replNum] [-p physicalPath] [-R resource] [-X restartFile] localSrcFile|localSrcDir ... destDataObj|destColl


**Store a file into iRODS.** If the destination data-object or collection are not provided, the current irods directory and the input file name are used. The -X option specifies that the restart option is on and the restartFile input specifies a local file that contains the restart info. If the restartFile does not exist, it will be created and used for recording subsequent restart info. If it exists and is not empty, the restart info contained in this file will be used for restarting the operation. Note that the restart operation only works for uploading directories and the path input must be identical to the one that generated the restart file.
$ iput -h
Options are:

- **-f** force - write data-object even if it exists already; overwrite it
- **-k** checksum - calculate a checksum on the data
- **-K** verify checksum - calculate and verify the checksum on the data
- **-N** numThreads - the number of transfer threads to use. A value of 0 means no threading. By default (-N option not used) the server decides the number of threads to use.
- **-R** resource - specifies the resource to store to. This can be specified in your environment or via a rule set up by the administrator.
- **-r** recursive - store the whole subdirectory
- **-v** verbose
- **-V** Very verbose
- **-X** restartFile - specifies that the restart option is on and the restartFile input specifies a local file that contains the restart info.
- **-h** this help
Put a File into the iRODS Collection

Class Exercise

$ cd /Applications/iRODS/clients/icommands/src

$ iput icd.c

$ ils -l

/tempZone/home/user1:

user1 0 demoResc 4427 2008-01-23.09:21 & icd.c
C- /tempZone/home/user1/looptest
C- /tempZone/home/user1/nvo
C- /tempZone/home/user1/tg
Resource Group

Class Exercise

iadmin - interactive mode
lr - list resources
lr demoResc - list demoResc
h mkresc - list options
  mkresc demo2Resc 'unix file system' archive
  '140.109.127.17' /Applications/iRODS/Vault2
lr - list resources
atrg dr demoResc - add resource to group
atrg dr demo2Resc - add resource to group
lrg dr - list resource group
iadmin
Class Exercise

• Access irods wiki at
  http://irods.sdsc.edu

• Search for “resource group”
  • Logical aggregation of storage resources

• Read irepl page
  • What happens when files are replicated to a resource group?
Create Storage Resources

$ iadmin mkresc nvoReplResc 'unix file system' archive '140.109.127.17' /Applications/iRODS/Vaultnvo

$ iadmin mkresc tgReplResc 'unix file system' archive '140.109.127.17' /Applications/iRODS/Vaulttg
List storage resources

$ ilsresc

demo2Resc
demoResc
nvoReplResc
tgReplResc
dr (resource group)
Transcontinental Persistent Archive Prototype

- Distributed Data Management Concepts
  - Data virtualization
    - Storage system independence
  - Trust virtualization
    - Administration independence
- Risk mitigation
  - Federation of multiple independent data grids
    - Operation independence
National Archives and Records Administration
Transcontinental Persistent Archive Prototype

Federation of Seven
Independent Data Grids

Extensible Environment, can federate with additional research and education sites. Each data grid uses different vendor products.
Data Management Challenges

- **Authenticity**
  - Manage descriptive metadata for each file
  - Manage access controls
  - Manage consistent updates to administrative metadata

- **Integrity**
  - Manage checksums
  - Replicate files
  - Synchronize replicas
  - Federate data grids

- **Infrastructure independence**
  - Manage collection properties
  - Manage interactions with storage systems
  - Manage distributed data
Digital Preservation

• Preservation is communication with the future
  • How do we migrate records onto new technology (information syntax, encoding format, storage infrastructure, access protocols)?
  • SRB - Storage Resource Broker data grid provides the interoperability mechanisms needed to manage multiple versions of technology
• Preservation manages communication from the past
  • What information do we need from the past to make assertions about preservation assessment criteria (authenticity, integrity, chain of custody)?
  • iRODS - integrated Rule-Oriented Data System
Data Grids

• SRB - Storage Resource Broker
  • Persistent naming of distributed data
  • Management of data stored in multiple types of storage systems
  • Organization of data as a shared collection with descriptive metadata, access controls, audit trails

• iRODS - integrated Rule-Oriented Data System
  • Rules control execution of remote micro-services
  • Manage persistent state information
  • Validate assertions about collection
  • Automate execution of management policies
Preservation Models

• Diplomatics (InterPARES)
  • Authenticity of records asserted by submitting institution
  • Records preserved forever

• Preservation lifecycle (NARA)
  • Arrangement / Hierarchical metadata - Record Group, Record series, Folder, Item, Object
  • Archival information packages (AIP)

• Continuum (NSDL)
  • Preservation within context of active records (active data grid)

• Digital library (DSpace)
  • Digital library standards for arrangement and description (METS, OAI-PMH)
InterPARES - Diplomatics

- Authenticity - maintain links to metadata for:
  - Date record is made
  - Date record is transmitted
  - Date record is received
  - Date record is set aside [i.e. filed]
  - Name of author (person or organization issuing the record)
  - Name of addressee (person or organization for whom the record is intended)
  - Name of writer (entity responsible for the articulation of the record’s content)
  - Name of originator (electronic address from which record is sent)
  - Name of recipient(s) (person or organization to whom the record is sent)
  - Name of creator (entity in whose archival fonds the record exists)
  - Name of action or matter (the activity for which the record is created)
  - Name of documentary form (e.g. E-mail, report, memo)
  - Identification of digital components
  - Identification of attachments (e.g. digital signature)
  - Archival bond (e.g. classification code)
InterPARES - Diplomatics

- Integrity - maintain links to metadata for
  - Name(s) of the handling office / officer
  - Name of office of primary responsibility for keeping the record
  - Annotations or comments
  - Actions carried out on the record
  - Technical modifications due to transformative migration
  - Validation
Preservation Rules

- **Authenticity**
  - Rules that quantify required descriptive metadata
  - Rules that verify descriptive metadata is linked to records
  - Rules that govern creation of AIPs

- **Integrity**
  - Rules that verify records have not been corrupted
  - Rules that manage replicas
  - Rules that recover from corruption instances
  - Rules that manage data distribution

- **Chain of custody**
  - Persistent identifiers for archivists, records, storage
  - Rules to verify application of access controls
  - Rules to track storage location of records
Integrity Challenges

- Data grids manage shared collections that are distributed across multiple storage systems and institutions
  - Data grids are responsible for providing recovery mechanisms for all errors that occur in the distributed environment
  - The number of observed problems is proportional to the size of the collections
Integrity Mechanisms

Class Exercise

• $ irepl -h


• Replicate a file in iRODS to another storage resource.

$ irepl -R demo2Resc foo1

$ ils -l

tempZone/home/user1:

user1 0 demoResc 4585 2007-08-30.14:33 & foo1
user1 1 demo2Resc 4585 2007-09-18.17:36 & foo1
Integrity Mechanisms

- `$ irsync -h`

- Synchronize the data between a local copy (local file system) and the copy stored in iRODS or between two iRODS copies. The command can be in one of the three modes:
  - synchronization of data from the client's local file system to iRODS,
  - from iRODS to the local file system,
  - from one iRODS path to another iRODS path.

- The mode is determined by the way the `sourceFile|sourceDirectory` and `targetFile|targetDirectory` are specified.
  - Files and directories prepended with 'i:' are iRODS files and collections.
  - Local files and directories are specified without any prependage.
Integrity Mechanisms

- `irsync -r foo1 i:foo2`
  - synchronizes recursively the data from the local directory `foo1` to the iRODS collection `foo2`

- `irsync -r i:foo1 foo2`
  - synchronizes recursively the data from the iRODS collection `foo1` to the local directory `foo2`.

- `irsync -r i:foo1 i:foo2`
  - synchronizes recursively the data from the iRODS collection `foo1` to another iRODS collection `foo2`.

- Checksums are used to determine whether a file should be synchronized.
Integrity Mechanisms

• `$ ichksum -h`
• Usage : `ichksum [-harvV] [-K|f] [-n replNum] dataObj|collection`
• Checksum one or more data-object or collection from iRODS space.
• Options are:
  • `-f` force checksum data-objects even if a checksum already exists
  • `-a` checksum all replica.
  • `-K` verify the checksum value in icat. If the checksum value does not exist, compute and register one.
  • `-n` replNum - the replica to checksum; if not specified checksum all replicas
  • `-r` recursive - checksum the whole subtree; the collection, all data-objects in the collection, and any subcollections and sub-data-objects in the collection.
Class Exercise - HELP.looptest

- Make two test collections, and load files from your system
  - mkdir loopTest
  - mkdir loopTest2
  - cd loopTest
  - iput ../src/ipwd.c
  - iput ../src/iquest.c
  - iput ../src/ils.c
  - ils -l
$ irule -h

Usage: irule [--test] [-v] rule inputParam outParamDesc

Submit a user defined rule to be executed by an irods server. The first form requires 3 inputs:

1) rule - This the rule to be executed.
2) inputParam - The input parameters for the rule are specified here. If there is no input, a string containing "null" must be specified.
3) outParamDesc - Description for the set of output parameters to be returned. If there is no output, a string containing "null" must be specified.
iRULE Command

Usage: irule [--test] [-v] [-l] -F inputFile [prompt | arg_1 arg_2 ...]

The second form reads the rule and arguments from the file: inputFile

- The first (non-comment) line is the rule. The remaining arguments are interpreted as input arguments for the rule.
- If prompt is the first remaining argument, the user will be prompted for values. The current value will be shown and used if the user just presses return.

- Otherwise, the arguments are interpreted in two ways
  - In the first way, the arguments have "label=value" format and only those given label-value pairs are replaced and other pairs are taken from the inputFile. All labels start with *.
  - Alternatively, one can give all arguments as inputs without any labels. In such a case the keyword default can be used to use the inputFile value. Use \ as the first letter in an argument as an escape.
iRULE Command

- The inputFile should contain 3 lines, the first line specifies the rule, the second line the input arguments as label=value pairs separated by % and the third line contains output parameters as labels again separated by %. If % is needed in an input value use ℅.%

- A value of an input argument can be $. In such a case the user will be prompted. One can provide a default value by giving it right after the $. In such a case, the value will be shown and used if the user presses return without giving a value. The input or the output line can be just be the word null if no input or output is needed.

- An example of the input is given in the file:
  - clients/icommands/test/ruleInp1

- In either form, the 'rule' is either a rule name or a rule definition (which may be a complete rule or a subset).

- To view the output (outParamDesc), use the -v option.

- See ruleInp1 for an example outParamDesc.
iRULE Command

- Options are:
  - --test enable test mode so that the micro-services are not executed, instead a loopback is performed
  - -F inputFile - read the file for the input
  - -l list file if -F option is used
  - -v verbose
  - -h this help
listColl.ir Rule

```
$cd /applications/iRods/clients/icommands/test
$vi listColl.ir
myTestRule | | acGetIcatResults(*Action,*Condition,*B)
  ##forEachExec(*B, msiPrintKeyValPair(stdout,*B)
  ##writeLine(stdout,*K),nop) | nop##nop
*Action=list%*Condition= COLL_NAME = '/tempZone/home/rods/loopTest'  
 '------%*K=--------
HAHAHAHAHAH---------------
*Action%*Condition%ruleExecOut
```
/* LISTING AND CHECKSUM */
cd /Applications/iRODS/clients/icomm/commands/bin
irule -F ./test/listCollir
ichkszum -r .
irule -F ./test/showicatchkszumCollir
/* use the following to change a file under iRODS
iquest "select DATA_PATH where DATA_NAME = 'iquest.c'"
vi
  **/modify file
irule -F ./test/verifychkszumCollir.ir
irule -F ./test/forcechkszumCollir.ir
Types of Risk

- **Media failure**
  - Replicate data onto multiple media
- **Vendor specific systemic errors**
  - Replicate data onto multiple vendor products
- **Operational error**
  - Replicate data onto a second administrative domain
- **Natural disaster**
  - Replicate data to a geographically remote site
- **Malicious user**
  - Replicate data to a deep archive
How Many Replicas

• Three sites minimize risk
  • Primary site
    • Supports interactive user access to data
  • Secondary site
    • Supports interactive user access when first site is down
    • Provides 2nd media copy, located at a remote site, uses different vendor product, independent administrative procedures
  • Deep archive
    • Provides 3rd media copy, staging environment for data ingestion, no user access
Data Reliability

- Manage checksums
  - Verify integrity
  - Rule to verify checksums
- Synchronize replicas
  - Verify consistency between metadata and records in vault
  - Rule to verify presence of required metadata
- Federate data grids
  - Synchronize metadata catalogs
$ cd /Applications/iRODS/server/config/reConfigs
Edit core.irb

To the rule “acSetRescSchemeForCreate” add a random sort after the default resource specification

    acSetRescSchemeForCreate||msiSetDefaultResc(demoResc, null)##msiSetRescSortScheme(random)|nop##nop

Make a subdirectory with a few small files
    mkdir d1
    ls > d1/foo1 (etc)
    iput -r -R dr d1 /* are using a resource group
    ils -l d1
    irm -r d1
Rule-based Data Management

- Map from management policies to rules controlling execution of remote micro-services
- Manage persistent state information for results of each micro-service execution
- Support an additional three logical name spaces
  - Rules
  - Micro-services
  - Persistent state information
- Constitutes representation information for preservation environments
Micro-Services

• Challenge is that storage systems do not provide required sophisticated operations
  • Have “minimal” set of standard operations that are performed at the storage system
  • Have actions required by clients such as replication, metadata extraction
  • Create standard micro-services that serve aggregate storage operations into modules that can be aggregated to implement desired client actions.
Micro-service Classes

- Test micro-services
- System micro-services
- Workflow micro-services
- System micro-services
- User micro-services called by client
- iCAT micro-services
- User micro-services invoked by “irule”
- Image manipulation micro-services
Example Micro-Services
http://irods.sdsc.edu/index.php/List_of_Micro-Services

- **Workflow Services:**
  - **nop, null** - no action
  - **cut** - not to retry any other applicable rules for this action
  - **succeed** - succeed immediately
  - **fail** - fail immediately - recovery and retries are possible
  - **msiGoodFailure** - useful when you want to fail but no recovery initiated.
  - **msiNullAction** - same as nop
  - **whileExec** - while loop over result set
  - **forExec** - for loop over result set
Example Micro-Services
http://irods.sdsc.edu/index.php/List_of_Micro-Services

- System Micro Services - Can only be called by the server process.
- msiSetDefaultResc - set the default resource
- msiSetNoDirectRescInp - sets a list of resources that cannot be used by a normal user directly.
- msiSetRescSortScheme - set the scheme for selecting the best resource to use
- msiSetMultiReplPerResc - sets the number of copies per resource to unlimited
- msiSetDataObjPreferredResc - if the data has multiple copies, specify the preferred copy to use
- msiSetDataObjAvoidResc - specify the copy to avoid
- msiSortDataObj - Sort the replica randomly when choosing which copy to use
- msiSetNumThreads - specify the parameters for determining the number of threads to use for data transfer.
- msiSysChksumDataObj - checksum a data object.
- msiSysReplDataObj - replicate a data object.
Example Micro-Services
http://irods.sdsc.edu/index.php/List_of_Micro-Services

- msiStageDataObj - stage the data object to the specified resource before operation.
- msiNoChkFilePathPerm - Do not check file path permission when registering
- msiNoTrashCan - Set the policy to no trash can.
- msiSetPublicUserOpr - Sets a list of operations that can be performed by the user "public".
- msiSetGraftPathScheme - Set the scheme for composing the physical path in the vault to GRAFT_PATH.
- msiSetRandomScheme - set the scheme for composing the physical path in the vault to RANDOM.
- msiCheckHostAccessControl - Set the access control policy.
- msiDeleteDisallowed - Set the policy for determining certain data cannot be deleted.
- msiSetResource() - sets the resource from default
- msiSetResourceList - get a resource based on conditions
- msiSetDataTypeFromExt() - get data type based on file name extension
• Classes of rules
  • Internal rules used to maintain consistency between operations and persistent state information
  • Administrator controlled rules that are automatically invoked
  • User executable rules
Rules

• Rule execution
  • Atomic rules - executed on each operation invoked by a client
  • Deferred rules - executed at a future time
  • Periodic rules - executed to validate assessment criteria and enforce desired properties
iRODS Rule Syntax

- Event | Condition | Action-set | Recovery-set
  - Event - triggered by operation or queued rule
  - Condition - composed by tests on any attributes in the persistent state information
  - Action-set - composed from both micro-services and rules
  - Recovery-set - used to ensure transaction semantics and consistent state information
iRODS Rule Sample

$ irule -F showcore.ir

5  core.acCreateUser
   {
      msiCreateUser     [msi.Rollback]
      acCreateDefaultCollections [msi.Rollback]
      msiCommit
   }

7  core.acCreateDefaultCollections
   {
      acCreateUserZoneCollections
   }

8  core.acCreateUserZoneCollections
   {
      acCreateCollByAdmin(/$rodsZoneProxy/home,$otherUserName)
      acCreateCollByAdmin(/$rodsZoneProxy/trash/home,$otherUserName)
   }

9  core.acCreateCollByAdmin(*parColl,*childColl)
   {
      msiCreateCollByAdmin(*parColl,*childColl)
   }
iCommands

- **iinit** initialize access
- **imkdir** *directory* make directory
- **ils** list files
- **ilsresc** list storage resources
- **iput** *directory file* put file into iRODS
- **iget** file get file from iRODS
- **imeta -h** list metadata options
Metadata Manipulation

Class Exercise

- $ imeta -h
  - Usage: imeta [-vVh] [command]
  - Commands are:
    - add -d|C|R|u Name AttName AttValue [AttUnits] (Add new AVU triplet)
    - rm -d|C|R|u Name AttName AttValue [AttUnits] (Remove AVU)
    - rmw -d|C|R|u Name AttName AttValue [AttUnits] (Remove AVU, use Wildcards)
    - ls -d|C|R|u Name [AttName] (List existing AVUs for item Name)
    - lsw -d|C|R|u Name [AttName] (List existing AVUs, use Wildcards)
    - qu -d|C|R|u AttName Op AttVal (Query objects with matching AVUs)
    - cp -d|C|R|u -d|C|R|u Name1 Name2 (Copy AVUs from item Name1 to Name2)

Metadata attribute-value-units triplets (AVUs) consist of an Attribute-Name, Attribute-Value, and an optional Attribute-Units. They can be added via the 'add' command and then queried to find matching objects.

For each command, -d, -C, -R or -u is used to specify which type of object to work with: dataobjs (iRODS files), collections, resources, or users. (Within imeta -c and -r can be used, but -C and -R are the iRODS standard options for collections and resources.)
Metadata Manipulation

Class Exercise

- `$ imeta add -d foo1 Genealogy Moore`
- `$ imeta add -d foo1 "number of persons" 175,143`
- `$ imetals -d foo1`

AVUs defined for dataObj foo1:

- attribute: Genealogy
- value: Moore
- units:

- attribute: number of persons
- value: 175143
- units:
Trash
Class Exercise

• `irm` - transfers file to the trash
• Trash collection is located at
  • `/tempZone/trash`
• Your directory structure is replicated as files are removed
  • `irm foo1`
  • `/tempZone/trash/user1/foo1`
• `irmtrash` removes files from trash
User Level Rules

• irule -F *rulename* Execute your rule

• Rules
  • showCore.ir list current rule base
  • listColl.ir list checksums
  • verifychksumColl.ir verify checksums
  • forcechksumColl.ir update checksums
  • replColl.ir replicate collection
Checksum Verification Example

$ more ../test/listColl.ir

First line:
myTestRule | | acGetIcatResults(*Action,*Condition,*B)##
forEachExec(*B,msiPrintKeyValPair(stdout,*B) ##
writeLine(stdout,*K),nop) |  nop ## nop

Second Line:
*Action=list%*Condition= COLL_NAME =
'/tempZone/home/rods/loopTest'%*K=--------FILE--------

Third line:
*Action%*Condition%ruleExecOut
$ irule -F showcore.ir

0  core.acPostProcForPut
   IF ($objPath like /tempZone/home/rods/nvo/*) {
      msiSysReplDataObj(nvoReplResc,null)
   }

1  core.acPostProcForPut
   IF ($objPath like /tempZone/home/rods/tg/*) {
      delayExec(<PLUSET>1m</PLUSET>,msiSysReplDataObj(tgReplResc,null),nop)
   }

2  core.acPostProcForPut
   IF ($objPath like *.mdf) {
      msiLoadMetadataFromFile [msiRollback]
   }
Test Replication
Class Exercise

• more irodsdemo.txt examples, create another resource

  iadmin mkresc demo3Resc 'unix file system' archive '140.109.127.17' /Applications/iRODS/Vault3

  ../../../server/bin/stop.pl
  ../../../server/bin/start.pl
Test Replication
Class Exercise

• imkdir nvo
• imkdir tg
• ils -l nvo
• iput -R demoResc ../src/icd.c nvo
• ils -l nvo

• How is this different?
Test Replication
Class Exercise

- ils -l tg
- iput -R demoResc ../src/icd.c tg
- ils -l tg

- How is this different?
- lqstat -l
  - Check that the second copy is made
iRODS Rich Web Client

• Use web browser to access a remote iRODS shared data collection
  • **https://rt.sdsc.edu:8443/irods/index.php**
  • IE, Firefox, Safari browser
    • Host/IP   rt.sdsc.edu
    • Port   1247
    • Username demoUser
    • Password demo
Web Client Collection Upload

- Click the 'Sign-on' button, and user should be redirected to his home collection '/tempZone/home/demoUser'
- Click on the 'new' button, and try to create a new collection 'upload'
  - Double click on the newly created collection 'upload'
- Click on the 'upload' button, and pick 'Single file'
  - Upload a image file with that dialog.
  - The file should appear after the uploading.
- Click the 'upload -> files and collections' button, the applet should load, and it will take a while (4-7 sec)
  - Drag and drop files/directories to the box inside applet.
  - Click 'upload' on the bottom of the applet, uploading will start.
- Close the applet layer.
  - The newly uploaded files/directories will appear after refreshing the current directory either by click on the refresh button on the bottom of the grid, or refresh the web page.
Web Client Image Manipulation

• Double click on a image file (jpg, png, or bmp) to bring up the file viewer dialog.
  • Click on the 'Open' button to open the image file with a new browser window
  • Close the new window and file viewer dialog
• **Click on the 'More -> metadata', to bring up the metadata dialog**
  • Add/modify/delete some random metadata, similar to spreadsheet operations, and click 'save', note that the metadata name must be unique.
  • Close the metadata dialog
• **Click on the 'delete' button, to delete the file.**
  • Type a partial file name in the quick search box, located at top right corner, and press enter. The partial name is case sensitive. For instance, if there is a file named 'bar_foo_bar.txt', a search for the string 'foo' will return the file. The result files are also clickable.
Web Client

- Click the 'v' button right next to the quick search box, this will bring up the advanced search dialog.
  - Fill in some metadata defined earlier and combine with some system attribute. It should return the files that match the search criteria.
- Click on the 'Sign Out' link on top right corner of the screen.
  - Only files are searchable. Collection search is not yet implemented, as a new server API is needed.
  - Replication is still under development.
iRODS describes the structured information required by each micro-service.

iRODS provides a standard structure for sending information over the network to the server and the client:

- rEi - ruleExecInfo structure

iRODS provides a structured information interface for retrieving information from a remote information resource.
Standard Micro-services

- Format specific data parsing
- Schema based input
- Schema based output
- Generate a DOI
- Shibboleth & GSI virtual organization
  - Map from virtual organization to the groups/roles within iRODS
- Shibboleth on Ajax rich web client
- High water marks - automated backup
- SRB workspace to compound object for publication in Fedora
Micro-services

- Flexible small file handling for network transport
- File transfer service - GridFTP/gLite
- Arbitrary transfer protocol - UDT, message bus
- End-to-end encryption - patient confidentiality
irule Examples

• Invocation and chaining of remote web services
• `getObjPositionByName.ir`
  • Accesses a sky catalog, issues a request to convert from object name to object location
    myTestRule||msiObjByName(*objName,*RA,*DEC,*TYPE)|nop
    *objName=$m100
    *objName%*RA%*DEC%*TYPE

• `getCutOutByPosition.ir`
  • Accesses a sky survey and retrieves an image cutout

• `getCutOutByObjName.ir`
  • Accesses the sky catalog, then gets the image cutout and registers the cutout into an iRODS collection
Delayed Execution

- Delayed rule which is executed every 6 minutes
  - starting 1 minute after its submission:

  - `actestMonPerf||delayExec(<PLUSET>1m</PLUSET><EF>6m</EF>, msiServerMonPerf(default, default),nop)|nop`

- `msiServerMonPerf` is executed every 6 minutes.

- `EA - execAddress` - host where the delayed execution needs to be performed
- `ET - execTime` - absolute time when it needs to be performed.
- `PLUSET - relExeTime` - relative to current time when it needs to execute
- `EF - execFreq` - frequency (in time widths) it needs to be performed.
Delayed Execution

The format for EF is quite rich:
* The EF value is of the format:
  * nnnnU <directive> where
  * nnnn is a number, and
  * U is the unit of the number (s-sec,m-min,h-hour,d-day,y-year),
* The <directive> can be for the form:
  * <empty-directive> - equal to REPEAT FOR EVER
  * REPEAT FOR EVER
  * REPEAT UNTIL SUCCESS
  * REPEAT nnnn TIMES - where nnnn is an integer
  * REPEAT UNTIL <time> - where <time> is of the time format supported by checkDateFormat function
  * REPEAT UNTIL SUCCESS OR UNTIL <time>
  * REPEAT UNTIL SUCCESS OR nnnn TIMES
  * DOUBLE FOR EVER
  * DOUBLE UNTIL SUCCESS - delay is doubled every time.
  * DOUBLE nnnn TIMES
  * DOUBLE UNTIL <time>
  * DOUBLE UNTIL SUCCESS OR UNTIL <time>
  * DOUBLE UNTIL SUCCESS OR nnnn TIMES
  * DOUBLE UNTIL SUCCESS UPTO <time>
irule Examples

- irodsdemo.txt
  - Lists examples for delayed execution

- HELP.looptest
  - Lists examples for checksums, copying, replicating, sending e-mail, purging files
  - Lists the irule tests that are validated for loops, remote execution, metadata extraction, web services
Topics on \( \mu \text{Services} \): What do we learn here?

- **Design of \( \mu \text{Services} \) for achieving a goal**
  - Extraction & Ingestion of template-identified metadata

- **Implementation of \( \mu \text{Services} \)**
  - How each module is coded to be \( \mu \)-compliant

- **Testing of \( \mu \text{Services} \)**
  - From the command line – no less.
  - A demo of all the services
  - as a workflow

You are in the driver seat
Hi! I am Micro!
Welcome to the World of μServices.
Hope you Enjoy it!
Problem Statement:

How to

Extract Metadata Attr-Value pairs from one file
Based on a Template defined in a second file, and
Associate the metadata to a third file?

Problem Break up:

Extract Metadata → Ingest Metadata → Commit Metadata

Sounds like 3 μServices!
Design: Lets look at the input files

**Template Files** are used to identify keyword/value pairs in a document

**Sample Tags:**

```
<PRETAG>Date: </PRETAG>SentDate<POSTTAG> </POSTTAG>
<PRETAG>From: </PRETAG>Sender<POSTTAG> </POSTTAG>
```

**Meaning:** Whatever is found between “Date :” and “ ” provides the “value” for the keyword: “SentDate”

**Metadata Files** provide the actual metadata that need to be ingested.

**Sample File:**

Date: Thu, 01 Feb 2007, 22:33:35 +000
From: adil hasan <a.hasan@rl.ac.uk>
To: Michael Wan <mwan@sdsc.edu>
Design: Extract Metadata

- **Open Template File**
- **Read Into Buffer 1**
- **Extract Tags**
- **Contains Triplets**
  
  \[<\text{PreTag}, \text{PostTag}, \text{Keyword}>\]

  *Example:*
  
  \[<\text{Date}, \text{SentDate}>\]

- **Open Metadata File**
- **Read Into Buffer 2**
- **Extract Key-Val Pairs**
- **Contains Pairs**
  
  \[<\text{Keyword}, \text{Value}>\]

  *Example:*
  
  \[<\text{SentDate}, \text{Thu, 01 Feb 2007, 22:33:35 +000}>\]
Instead of writing a \(\mu\)Service for file-metadata ingestion only, I just designed a \(\mu\)Service that can be applied to any iRODS object (data, collection, resource, user or token or metadata).

**Question**: How to convert a C-function into a \(\mu\)Service?
Design: Epilogue

Open TF → Read Into Buffer1 → Extract Tags

Open MF → Read Into Buffer2 → Extract KVPairs

I Commit! So iCAT!

Commit → Ingest Metadata By Object Type → Find Object Type

Key-Val Pairs Structure

Template File

Metadata File

So it is not just 3 μServices

Target ObjName

UCSD
Implementation of $\mu$Services
**Implementation: Prologue**

Four Easy Steps:

1) Define the signature of the \( \mu \)Service
2) Register the \( \mu \)Service as an invokable method by the rule engine
3) Create the \( \mu \)Service
   - This may need other function calls (new and old)
4) Describe the \( \mu \)Service

We will look at two Examples:

- “FindObjectType” \( \mu \)Service: `msiGetObjType`
- “Extract Tag” \( \mu \)Service: `msiReadMDTemplateIntoTagStruct`
Implementation: Signature Definitions

All μServices have only two types of parameters
- Params 1…(n-1) are of the type `msParam_t`
- Param n is of the type `ruleExecInfo_t`

`msParam_t` is defined as:

```c
typedef struct MsParam {
    char *label;
    char *type;
    void *inOutStruct;
    bytesBuf_t *inpOutBuf;
} msParam_t;
```

`ruleExecInfo_t` is the “white board” used for passing session-oriented parameters that can be used by the Rule Engine and the micro-services.

```
int msiGetObjType (msiParam_t *objParam, msiParam_t *typeParam, ruleExecInfo_t *rei);
int msiReadMDTemplateIntoTagStruct (msiParam_t *bufParam, msiParam_t *tagParam, ruleExecInfo_t *rei);
```
The Rule Engine only executes μServices that are enumerated in the list structure:

```c
microsdef_t MicrosTable[ ] = {} ;
```

(can be found in the file server/include/reAction.h)

For the μService

```c
int msiGetObjType (msiParam_t *objParam, msiParam_t *typeParam, ruleExecInfo_t *rei);
```

We add the following to the ActionTable

```c
{“msiGetObjType”,  2, (funcPtr) msiGetObjType}
```
Implementation: Registration “ExtractTag”

For the $\mu$Service

```c
int msiReadMDTemplateIntoTagStruct (msiParam_t *bufParam, msiParam_t *tagParam, ruleExecInfo_t *rei);
```

We add the following to the ActionTable

```c
{“msiReadMDTemplateIntoTagStruct”,
  2, (funcPtr) msiReadMDTemplateIntoTagStruct}
```
Implementation:  Creation “FindObjectType”

Here we program the code for the μService

```c
int msiGetObjType (msiParam_t *objParam,
                   msiParam_t *typeParam, ruleExecInfo_t *rei)
{
    char*   objName;
    char    objType[MAX_NAME_LEN];
    int     i;
    RE_TEST_MACRO("Looping back on msiGetObjType");
    if (strcmp(objParam->type, STR_MS_T) != 0)
        return(USER_PARAM_TYPE_ERROR);
    objName = (char *) objParam->inpOutStruct;
    i = getObjType (rei->rsComm, objName, objType);
    if (i < 0)   return(i);
    typeParam->inOutStruct = (char *) strdup(objType);
    typeParam->type = (char *) strdup(STR_MS_T);
    return(0);
}
```

- **Type Checking**
- **Internal Function**
  that is used for finding types of Objects. This routine makes calls to iCAT to find the type of the Object
- **Returning value**
  being malloc’d into Param Structure and type-cast properly

Needed for Loop Back Testing of Workflow and Rules
We want to provide enough material for users to call the μService and for a program to identify it automatically in the future.

/**
 * \fn   GetObjType
 * \author Arcot Rajasekar
 * \date   2007-02-01
 * \brief   this function finds from the iCat the type of a given object
 * \param[in]    objParam is a msParam of type STR_MS_T
 * \param[out]   typeParam is a msParam of type STR_MS_T
 * \return integer
 * \retval 0 on success
 * \retval USER_PARAM_TYP_ERROR when input param does not match the type
 * \retval from getObjType
 * \sa   getObjType
 * \post
 * \pre
 * \bug   no known bugs
 **/
Implementation: Creation “Extract Tag”

Here we program the code for the μService

```c
int msiReadMDTemplateIntoTagStruct (msiParam_t *bufParam,  
    msiParam_t *tagParam, ruleExecInfo_t *rei)  
{
    bytesBuf_t   *tmplObjBuf;
    tagStruct_t   *tagValues;
    /* other internal variables are defined here */

    RE_TEST_MACRO("Looping back on msiReadMDTemplateIntoTagStruct");

    if (strcmp(bufParam->type, BUF_LEN_MS_T) != 0 || bufParam->inpOutBuf == NULL)
        return(USER_PARAM_TYPE_ERROR);

    tmplObjBuf = (bytesBuf_t *) bufParam->inpOutBuf;
    tagValues = (tagStruct_t *) mallocAndZero(sizeof(tagStruct_t));

    /* the main code segment that reads the buffer and identifies the  
       <preTag, KeyWord, postTag>  triples goes in here. The triplets are stored in  
       tagValues. */

    if (tagValues->len == 0)  { free(tagValues ); return(NO_VALUES_FOUND); }  
    tagParam->inOutStruct = (void *) tagValues;
    tagParam->type = (char *) strdup(TagStruct_MS_T);
    return(0);
}
```
**Implementation: Describe “Extract Tag”**

We want to provide enough material for users to call the μService and for a program to identify it automatically in the future.

```c
/**
 * \fn msiReadMDTemplateIntoTagStruct
 * \author Arcot Rajasekar
 * \date   2007-02-01
 * \brief   this function parses a buffer containing a template-style file and stores the tags in a tag structure.
 * \note    the template buffer should contain triplets be of the form
 * <PRETAG>re1</PRETAG>kw<POSTTAG>re2</POSTTAG>
 * re1 identifies the pre-string and re2 identifies the post-string, and any value between re1 and re2 in a metadata buffer can be associated with keyword kw.
 * \param[in] bufParam is a msParam of type BUF_MS_T
 * \param[out] tagParam is a msParam of type TagStruct_MS_T
 * \return integer
 * \retval 0 on success
 * \retval USER_PARAM_TYP_ERROR when input param don’t match the type
 * \retval INVALID_REGEXP if the tags are not correct
 * \retval NO_VALUES_FOUND if there are no tags identified
 * \retval from addTagStruct
 * \sa addTagStruct
 * \post
 * \pre
 * \bug no known bugs
/**
```
RECAP:

Define Function Signature → Register μService Mapping → Create Internal Function → Describe μService

Any Function can be easily converted into a μService ---- μCompliant. Except that.....

Important!! Implement recovery μService
Testing of μServices
Testing: Prologue

Client-side: the \textit{irule} command
  Create a workflow of $\mu$Services
  Test with the “loop” functionality
  Test with “verbose” functionality
  Test without these side-effects

Server-side:
  Create a rule out of the workflow, or
  Add the $\mu$Service to an existing rule

Client-side:
  Test the rule using the \textit{irule} command

Semantics Testing is under research:
Testing: Micros

msiDataObjOpen
  opens a iRODS File
msiDataObjRead
  reads an open iRODS File
msiReadMDTemplateIntoTagStruct
  reads Tag Info into Struct
msiExtractTemplateMDFromBuf
  gets MD using Tag Struct
msiGetObjType
  finds type of object
msiAssociateKeyValuePairsToObj
  ingests extracted metadata
msiCommit
  commit transaction in iCAT

openObj
readObj
getTagsForKV
getKVPairsUsingTags
findObjType
ingestBulkMD
commitIcat

External Aliases Help Application Developers and Users
Testing: Workflow Diagram

- **openObj**
  - Template File Name → TFileHndl
  - readObj → TBuffer
    - getTags ForKV → Tag Struct

- **openObj**
  - MetaData File Name → MDFileHndl
  - readObj → MBuffer
    - getKVPairs UsingTags → Key-Val Pairs Struct

- **Commit iCAT**
- **Clean Up**
- **Ingest BulkMD**

- **findObj**
  - ObjType → iRODS Object Name

Yellow values are inputs
Pretty Printed Listing of File "ruleInp5"

mDExtract | openObj( *A, *T_FD)##getSizeData(*A,*S)##
readObj( *T_FD, *S, *R1_BUF)##
getTagsForKV( *R1_BUF, *TSP)##
openObj( *B, *M_FD)##
readObj( *M_FD, 10000, *R2_BUF)##
getKVPairsUsingTags( *R2_BUF, *TSP, *KVP)##
findObjType( *C, *OTYP)##
ingestBulkMD( *KVP, *C, *OTYP)##
closeObj(*T_FD,*J1)##closeObj(*M_FD,*J2)##
commitIcat

*A=/tempZone/home/rods/Templates/mdTemplate1.txt%
*B=/tempZone/home/rods/test1.email%
*C=/tempZone/home/rods/test2.email
*R1_BUF%*TSP%*R2_BUF%*KVP%*A%*B%*C%*OTYP

How to run it: irule –v –F ruleInp5
Testing: Making a Rule

The rule is very similar to the workflow we had seen in the previous slide.

```
mDExtract(*A,*B,*C) || openObj(*A,*T_FD)##
readObj(*T_FD, 10000, *R1_BUF)##
getTagsForKV(*R1_BUF, *TSP)##
openObj(*B,*M_FD)##
readObj(*M_FD, 10000, *R2_BUF)##
getKVPairsUsingTags(*R2_BUF, *TSP, *KVP)##
findObjType(*C,*OTYP)##
ingestBulkMD(*KVP, *C, *OTYP)##
closeObj(*T_FD,*J1)##closeObj(*M_FD,*J2)##
commitIcat
```

|closeObj(*T_FD)##nop##
|recover_getTagsForKV(*R1_BUF, *TSP)##
closeObj(*M_FD)##nop##
|recover_getKVPairsUsingTags(*R2_BUF, *TSP, *KVP)##
nop##
|recover_ingestBulkMD(*KVP, *C, *OTYP)##
nop##nop##rollbackIcat

Recovery Section

No Conditions are here
Delaying a μService

One can delay the execution of any μService either in the irule execution or in a rule at the server side.

Example:

The μService msiSysReplDataObj(*R) replicates an existing iRODS file.

In order to delay this, one can use:

delayExec(<PLUSET>2m</PLUSET>,msiSysReplDataObj( tgReplResc,null),nop)

In a rule this might be used as follows:

```
acPostProcessForPut | $objPath like /tmpZone/home/tg/* |
    delayExec(<PLUSET>2m</PLUSET>,msiSysReplDataObj( tgReplResc,null),nop)
    | nop
acPostProcessForPut | $objPath like /tmpZone/home/nvo/* |
    msiSysReplDataObj( nvoReplResc ) | nop
acPostProcessForPut | | nop | nop
```
Demonstration & Conclusion

Design of \( \mu \)Services for achieving a goal
- Extraction & Ingestion of template-identified metadata

Implementation of \( \mu \)Services
- How each module is coded to be \( \mu \)-compliant

Testing of \( \mu \)Services
- From the command line – no less.
- A demo of all the services as a workflow

Hope you Enjoyed it!
Any Questions!!
How to Communicate with a MicroService

MicroServices can access information in multiple ways:

- **Parameter passing**
  - Explicit number of arguments
  - Multiple types supported

- **Using a whiteboard architecture**
  - Implicit Information
  - ruleExecInfo Structure
  - Global Information used through the entire rule invocation

- **Out-of-band communication - Notification Service**
  - Using a message-passing system
  - Under implementation
  - Useful for parallel-invoked microServices
Parameter Passing -1

• Part of the MicroService Signature

```c
int findObjType ( msiParam_t *objInParam , msiParam_t *typeOutParam ,
ruleExecInfo_t *rei );
```

```c
int ingestBulkMD ( msiParam_t *objInParam, msiParam_t *typInParam,
msiParam_t *keyValuePairsInParam, ruleExecInfo *rei);
```

• All calling parameters follow the same structure

```c
typedef struct MsParam {
    char *label;       /* name of the parameter */
    char *type;        /* type of the parameter */
    void *inOutStruct; /* pointer to value/structure of the parameter */
    bytesBuf_t *inpOutBuf; /* optional buffer pointer for binary values */
} msParam_t;
```
Parameter Passing - 2

• A rule using the microServices (rei is implicit)

```c
void insertObjMetadata( *objName, *metaFileName)
{
    extractKVPairs( *metaFileName, *kvPairs) ::= null
    findObjType( *objName, *objType) ::= getGenericType(*objName,
              *objType)
    ingestBulkMD( *objName, *objType, *kvPairs) ::= rollback
}
```

• **How the rule works:**
  • Invoke the rule with objName and metaFileName values set in msParam_t* Structures
  • *kvPairs structure is populated by extractKVPairs microService
  • *objType structure is set by findObjType microService
  • ingestBulkMD uses the newly found values in kvPairs and objType, along with the input objName value.
  • When the rule is finished the new structures are freed.
  • All values are passed in and out as msParam_t* structures
WhiteBoard Architecture -1

- **ruleExecInfo (**rei**)
  - A large data structure that is passed when invoking a rule
  - Implicitly used throughout processing of the rule
  - MicroServices can access values/structs in the **rei** and also set values in the **rei** structure
  - The structure is defined in reGlobalsExtern.h and can be extended if necessary
  - Contains multiple important structures used in the iRODS data management:
    - **rsComm** - client-server communication structure
    - **doi** - dataObject information
    - **rescGrp** - resource (group) informations
    - **uoic** - client user information
    - and others ….
  - The rule invoking the function should set the proper values…
**WhiteBoard Architecture - 2**

- **$-variables**
  - Provides a way to logically use the values in the rei structure
  - Provides a mapping from a logical name to values in rei.
  - These mappings are defined in a configuration file:
    - objPath: rei->doi->objPath
    - rescName: rei->doi->rescName
    - userNameClient: rei->uoic->userName
  - One can use the variables in rules and microServices
    - assign($rescName, sdsc-samqfs ) /* assign is a microService*/
    - Condition checking: $objPath like /zone/home/sekar@sdsc/nvo/*
    - Parameter passing: findObjType($objName,*Type)
  - Assign values internally in the microServices:
    - rei->doi->rescName = strdup("sdsc-samfs"); /* directly set the value in the rei struct */
    - setVarValue("$rescName", rei, value); /* set the value using $variable name */
  - Access values internally in the microServices
    - valuePtr = rei->doi->rescName; /* get value directly from rei stuct */
    - getVarValue("$rescName",rei, &valuePtr); /* get value using $variable name */
## Persistent State Attribute Names

<table>
<thead>
<tr>
<th>Data attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA_ID</td>
<td>Unique identifier for a registered file</td>
</tr>
<tr>
<td>DATA_COLL_ID</td>
<td>Unique identifier for the collection</td>
</tr>
<tr>
<td>DATA_NAME</td>
<td>Logical name of the file in the data grid</td>
</tr>
<tr>
<td>DATA_REPL_NUM</td>
<td>Replication number of the file.</td>
</tr>
<tr>
<td>DATA_VERSION</td>
<td>Version of the file.</td>
</tr>
<tr>
<td>DATA_TYPE_NAME</td>
<td>Type of the file (.doc, .pdf, ?)</td>
</tr>
<tr>
<td>DATA_SIZE</td>
<td>Size of the file in bytes</td>
</tr>
<tr>
<td>DATA_RESC_GROUP_NAME</td>
<td>Group name of storage resources</td>
</tr>
<tr>
<td>DATA_RESC_NAME</td>
<td>Storage resource name for file storage</td>
</tr>
<tr>
<td>DATA_PATH</td>
<td>Physical path name of the file</td>
</tr>
<tr>
<td>DATA_OWNER_NAME</td>
<td>Owner of the file (USER_NAME)</td>
</tr>
<tr>
<td>DATA_OWNER_ZONE</td>
<td>Home zone of the owner (USER_ZONE)</td>
</tr>
<tr>
<td>DATA_REPL_STATUS</td>
<td>Status condition of a file (current, stale)</td>
</tr>
<tr>
<td>DATA_CHECKSUM</td>
<td>MD5 checksum of the file</td>
</tr>
<tr>
<td>DATA_EXPIRY</td>
<td>Retention date of the file</td>
</tr>
<tr>
<td>DATA_CREATE_TIME</td>
<td>Date of file registration</td>
</tr>
<tr>
<td>DATA_MODIFY_TIME</td>
<td>Last time the file was modified.</td>
</tr>
</tbody>
</table>
## Collection Attributes

<table>
<thead>
<tr>
<th>Collection attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLL_ID</td>
<td>Unique collection identifier</td>
</tr>
<tr>
<td>COLL_NAME</td>
<td>Collection name</td>
</tr>
<tr>
<td>COLL_PARENT_NAME</td>
<td>Name of the parent collection</td>
</tr>
<tr>
<td>COLL_OWNER_NAME</td>
<td>Name of the collection owner (USER_NAME)</td>
</tr>
<tr>
<td>COLL_OWNER_ZONE</td>
<td>Name of the home zone of the collection owner (USER_ZONE)</td>
</tr>
<tr>
<td>COLL_COMMENTS</td>
<td>Owner defined comments</td>
</tr>
<tr>
<td>COLL_CREATE_TIME</td>
<td>Date collection was created</td>
</tr>
<tr>
<td>COLL_MODIFY_TIME</td>
<td>Time collection was last modified</td>
</tr>
</tbody>
</table>
## Resource Attributes

<table>
<thead>
<tr>
<th>Resource attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESC_ID</td>
<td>Unique identifier for a storage resource within a zone</td>
</tr>
<tr>
<td>RESC_NAME</td>
<td>Name of the storage resource</td>
</tr>
<tr>
<td>RESC_ZONE_NAME</td>
<td>Name of the zone in which the resource is located</td>
</tr>
<tr>
<td>RESC_TYPE_NAME</td>
<td>Type of storage resource (unix-file-system)</td>
</tr>
<tr>
<td>RESC_CLASS_NAME</td>
<td>Class of storage resource (archival, permanent disk, cache, temporary disk)</td>
</tr>
<tr>
<td>RESC_LOC</td>
<td>Location of storage resource (IP internet address)</td>
</tr>
<tr>
<td>RESC_VAULT_PATH</td>
<td>Path name under which files are stored on resource</td>
</tr>
<tr>
<td>RESCCOMMENT</td>
<td>Data grid administrator comment on storage resource</td>
</tr>
<tr>
<td>RESC_CREATE_TIME</td>
<td>Date storage resource vault was created</td>
</tr>
<tr>
<td>RESC_MODIFY_TIME</td>
<td>Last time the storage resource vault was modified</td>
</tr>
</tbody>
</table>
Mounted Collection Interface

- Generic infrastructure for manipulating structured information
  - Storage System
    - Remote directory
  - Structured file
    - Archival Information Package
    - Tar file / XFDU / HDF5 / METS package / XAM
  - Database
    - Tabular information / BLOBS
  - Bulk operations
  - Registration
Structured Information Access

- MountCollCreate - create a new MountColl.
- MountCollOpen - open a MountColl file for query.
- MountCollRead - query for structures and directory structure.
- MountCollClose - close an opened MountColl.
- MCollSubCreate - create a structure in a MountColl.
- MCollSubOpen - open an existing structure in a MountColl.
- MCollSubRead - read the content of an opened structure.
- MCollSubWrite - write to an opened structure.
- MCollSubClose - close an opened structure.
- MCollSubunlink - delete an existing structure.
- MCollSubStat - get the status of an existing structure.
- MCollSubFstat - get the status of an opened structure.
- MCollSubLseek - lseek into an opened structure.
- MCollSubRename - rename an existing structure.
Management Policies

- Data distribution
- Instrument data production - object ring buffer with migration to an archive
  - Notification service
  - Message bus buffering to an archive
- Virtual Organization management of institutional repositories - hierarchical management policy (higher level policy that augments a local policy)
  - Policy hierarchy (approval flags)
Management Policies

• Analysis completion management
  • Sets of process status flags - control of processing pipeline from asynchronous sources

• Scheduling
  • Minimal execution rate required to meet a deadline
  • Resource limited scheduling

• Event and time dependent access control based on an approval flag

• Audit trails - verify actions performed on data
Theory of Data Management

- **Characterization**
  - Persistent name spaces
  - Operations that are performed upon the persistent name spaces
  - Changes to the persistent state information associated with each persistent name space that occur for each operation
  - Transformations that are made to the records on each operation

- **Completeness**
  - Set of operations is complete, enabling the decomposition of every management process onto the operation set.
  - Management policies are complete, enabling the validation of all assessment criteria.
  - Persistent state information is complete, enabling the validation of authenticity and integrity.

- **Assertion**
  - If the operations are reversible, then a future management environment can recreate a record in its original form, maintain authenticity and integrity, support access, and display the record.
  - Such a system would allow records to be migrated between independent implementations of management environments, while maintaining authenticity and integrity.
Preservation Projects

- 1999-2008: NARA - Transcontinental Persistent Archive Prototype
- 2000: NHPRC InterPARES I & II - International Preservation of Authentic Records in Electronic Systems
- 2002-2007: NSF NSDL - Web crawl preservation
- 2004: NHPRC - Persistent Archive Testbed
- 2004: UCSD Libraries - Image collection
- 2004: NARA - DSpace/SRB integration
- 2005: LC NDIIPP - CDL Digital Preservation Repository
- 2005: NSF/LC Digital Archiving project - UCSDtv “Conversations with History”
- 2006: IMLS - UCHRI data grid for humanities
- 2007: NSF Software Development Cyberinfrastructure
Digital Preservation

• Preservation manages communication from the past
  • What information do we need from the past to make assertions about preservation assessment criteria (authenticity, integrity, chain of custody)?

• RLG/NARA Trusted Repository Audit and Certification Criteria
RLG/NARA - TRAC Criteria

• Assessment categories
  • Organizational infrastructure
  • Digital Object Management
  • Technologies, Technical Infrastructure and Security

• Example criteria
  • B6.4 Repository has documented and implemented access policies (authorization rules, authentication requirements) consistent with deposit agreements for stored objects.
Classes of Assessment Criteria

- **Collection properties**
  - List properties of associated name spaces
  - Verify properties
  - Compare properties with assertions

- **Collection operations**
  - Transform file formats
  - Migrate data
  - Generate audit trails

- **Structured information**
  - Parse audit trails to generate compliance reports
  - Apply templates to extract information
  - Apply templates to format state information
Mapping to a Set of Rules

• List staff who have archivist execution permission on collection
• List all persons with access permissions on collection
• Analyze audit trails to verify identity of all persons accessing the data, and compare their roles with desired access controls
• Generate report listing all persons who accessed or applied archival functions on the collection
• Compare report with the deposition agreement
Persistent State Information

• Users
  • Allowed roles - archivist, management, owner
  • Training courses taken and dates
  • Involvement with a collection

• Resources
  • Audit log of all error incidents
  • Cost per TB of storage and date defined
  • Audit log of all upgrades
Persistent State Information

- Records
  - SIP compliance flag
  - Location of ingestion SIP
  - Original record ID
  - Master copy flag
  - Formal acceptance flag
  - Audit log of all applied micro-services by person and date
  - Format type
  - Size
  - Checksum and validation date
  - Replica locations
  - Provenance metadata
Persistent State Information

- Rules
  - Rule type
  - Version number
- Micro-services
  - Audit log of all changes
  - Version number
- Structured information
  - Template defining preservation attributes
  - Template for SIP format
  - Template for AIP format
  - Template to map from SIP template to AIP template
## RLG/NARA Assessment

- Are developing 105 rules that implement the TRAC assessment criteria

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>90</strong></td>
<td>Verify descriptive metadata and source against SIP template and set SIP compliance flag</td>
</tr>
<tr>
<td><strong>91</strong></td>
<td>Verify descriptive metadata against semantic term list</td>
</tr>
<tr>
<td><strong>92</strong></td>
<td>Verify status of metadata catalog backup (create a snapshot of metadata catalog)</td>
</tr>
<tr>
<td><strong>93</strong></td>
<td>Verify consistency of preservation metadata after hardware change or error</td>
</tr>
</tbody>
</table>
Federation Between Data Grids

Data Access Methods (Web Browser, DSpace, OAI-PMH)

Data Collection A

Data Grid
- Logical resource name space
- Logical user name space
- Logical file name space
- Logical rule name space
- Logical micro-service name
- Logical persistent state

Data Collection B

Data Grid
- Logical resource name space
- Logical user name space
- Logical file name space
- Logical rule name space
- Logical micro-service name
- Logical persistent state
Rule-based Federation

- Exchange files between data grids
  - Associate management policies with the exchanged files
  - Associate micro-services with the exchanged files
- Implication is that can specify redaction requirements on a file that is deposited in another data grid
iRODS Development

• NSF - SDCI grant “Adaptive Middleware for Community Shared Collections”
  • iRODS development, SRB maintenance
• NARA - Transcontinental Persistent Archive Prototype
  • Trusted repository assessment criteria
• NSF - Ocean Research Interactive Observatory Network (ORION)
  • Real-time sensor data stream management
• NSF - Temporal Dynamics of Learning Center data grid
  • Management of IRB approval
iRODS Development Status

• Production release is version 1.0
  • January 24, 2008

• International collaborations
  • SHAMAN - University of Liverpool
    • Sustaining Heritage Access through Multivalent ArchiviNg
  • UK e-Science data grid
  • IN2P3 in Lyon, France
  • DSpace policy management
Planned Development

- GSI support (1)
- Time-limited sessions via a one-way hash authentication
- Python Client library
- GUI Browser (AJAX in development)
- Driver for HPSS (in development)
- Driver for SAM-QFS
- Porting to additional versions of Unix/Linux
- Porting to Windows
- Support for MySQL as the metadata catalog
- API support packages based on existing mounted collection driver
- MCAT to ICAT migration tools (2)
- Extensible Metadata including Databases Access Interface (6)
- Zones/Federation (4)
- Auditing - mechanisms to record and track iRODS metadata changes
Development

- Shibboleth - collaboration with ASPIS