E-iRODS Composable Resources

Terrell Russell, Jason Coposky, Harry Johnson, Ray Idaszak, Charles Schmitt
Renaissance Computing Institute, University of North Carolina at Chapel Hill

renci
RESEARCH ENGAGEMENT INNOVATION

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Plugin Architecture

- Defines a common plugin interface
- Concentrates functionality into independently testable units
- Affords independent community development of new features external to the core
- Enables distribution of proprietary plugins in a binary-only format
- Affords the ability that plugins are dynamically available to a running server upon installation
- Includes a “ContextString” as flexible metadata which allows each instance of a plugin to be independently configured
- Enables multiple instances with different specific parameterization (e.g. multiple WOS, S3, etc. per E-iRODS server)

Resource Plugins

- Follow existing common iRODS file driver interface (fileOpen, fileClose, fileRead, fileWrite, etc.)
- Use the “ContextString” to define resource-specific metadata (e.g. S3 endpoint, WOS URL and port, path to local credential file)
- Capture complexity of resource behavior into reusable components
- Repackage existing functionality into plugins

E-iRODS Composable Resources

RENCI has developed composable resources for Enterprise iRODS (E-iRODS) which allow for shareable, flexible definitions of storage resources. Using a well-defined tree metaphor to describe composite resources provides insight into existing resources as well as a powerful tool for envisioning new resource configurations. Defined resources can be shared and iterated within the community easily as they are plugins, external from the E-iRODS core.

- Plugins
- Tree Metaphor
- Virtualization
- Extensible
- Community Driven

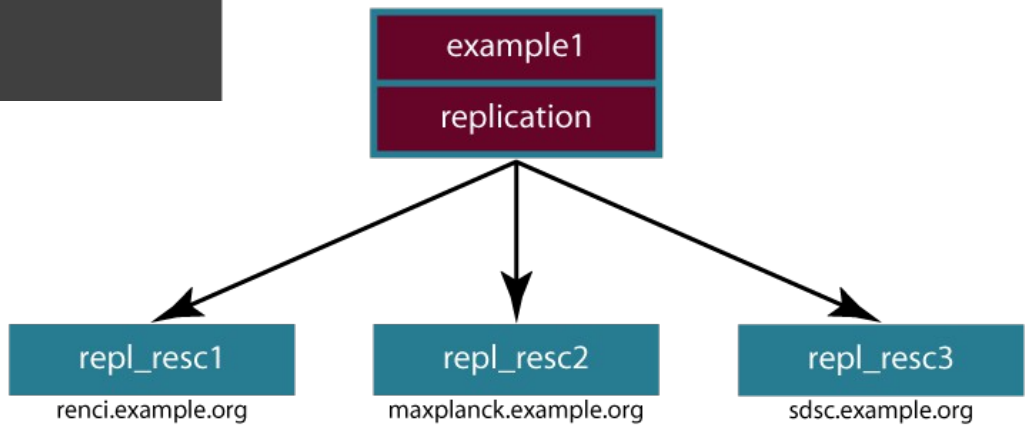
Example 1: Replicates Data Objects to three locations

Coordinating Resource – Strictly virtual (lives in the iCAT only)

It is the coordinating resource's responsibility to provide a "voting" mechanism for which replica is returned when the Data Object is requested by a user (i.e. iget).

This voting could be as simple as returning the first replica in the iCAT database or as complicated as keeping statistics on throughput or load or latency.

```
01 $ iadmin mkresc example1 replication
02 $ iadmin mkresc repl_resc1 "unix file system" renci.example.org:/Vault
03 $ iadmin mkresc repl_resc2 "unix file system" maxplanck.example.org:/Vault
04 $ iadmin mkresc repl_resc3 "unix file system" sdsc.example.org:/Vault
05 $ iadmin addchildtoresc example1 repl_resc1
06 $ iadmin addchildtoresc example1 repl_resc2
07 $ iadmin addchildtoresc example1 repl_resc3
```



Example 2: Tiered storage system with a dark archive

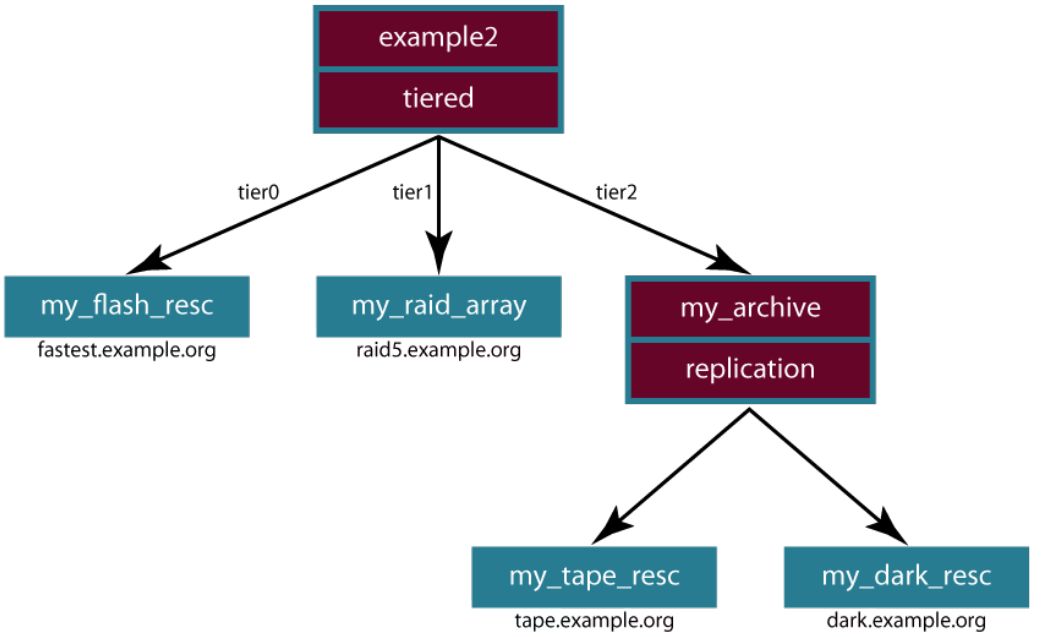
The relationship between a parent and child resource may also be defined with a ContextString.

Here, the three children of the tiered coordinating resource are ordered by speed with the fastest tier0 being a flash device, tier1 being a regular raid array of spinning disk, and a third tier2 being a replication coordinating resource with two children of its own.

The tier2 "my_archive" is replicating onto a tape device of some kind as well as a write-only space known as a dark archive.

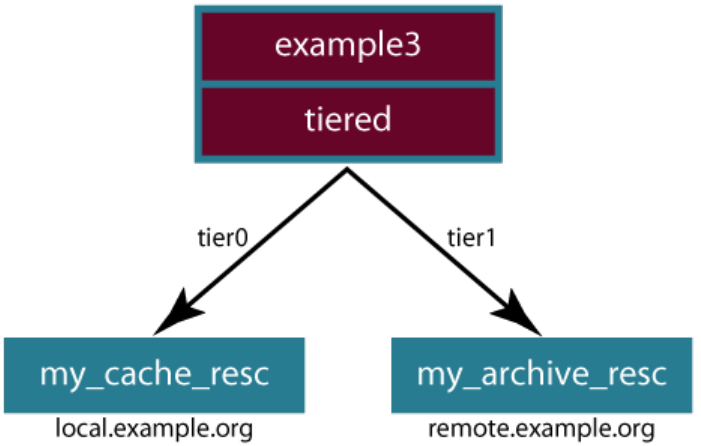
This "my_dark_resc" could have policy around it specifying that only admins can write, and no user can read. A dark archive would then always "vote" "no" requiring its parent to coordinate its own vote when responding to its own parent.

```
01 $ iadmin mkresc example2 tiered
02 $ iadmin mkresc my_flash_resc "unix file system" fastest.example.org:/Vault
03 $ iadmin mkresc my_raid_array "unix file system" raid5.example.org:/Vault
04 $ iadmin mkresc my_archive replication
05 $ iadmin mkresc my_tape_resc "unix file system" tape.example.org:/Vault
06 $ iadmin mkresc my_dark_resc "unix file system" dark.example.org:/Vault
07 $ iadmin addchildtoresc example2 my_flash_resc "tier0"
08 $ iadmin addchildtoresc example2 my_raid_array "tier1"
09 $ iadmin addchildtoresc example2 my_archive "tier2"
10 $ iadmin addchildtoresc my_archive my_tape_resc
11 $ iadmin addchildtoresc my_archive my_dark_resc
```



Example 3: Simulates legacy compound resource type

```
01 $ iadmin mkresc example3 tiered
02 $ iadmin mkresc my_cache_resc "unix file system" local.example.org:/Vault
03 $ iadmin mkresc my_archive_resc "unix file system" remote.example.org:/Vault
04 $ iadmin addchildtoresc example3 my_cache_resc "tier0"
05 $ iadmin addchildtoresc example3 my_archive_resc "tier1"
```



The local "my_cache_resc" would be consulted first for any read request that comes into "example3".

A write request would go into the cache resource first as well and then be replicated to its peer, the remote "my_archive_resc".

This example can duplicate the existing functionality of the compound resource as defined in community iRODS.

Thank You

Terrell Russell, Ph.D.
Renaissance Computing Institute (RENCI)
University of North Carolina at Chapel Hill