LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

IV. dCache Workshop
Bergische Universität Wuppertal
Wuppertal

April 13, 2010

Info Service, GLUE And Checksum Module

Christoph Anton Mitterer
christoph.anton.mitterer@lmu.de

Contents

This presentation consists of the following chapters:

I. Info Service
   Covers the information system provided by dCache.
II. GLUE
   Gives an overview of the GLUE schema and describes how to generate the corresponding data within dCache.
III. Checksum Module
   Describes the data integrity functionalities provided by dCache's Checksum Module.
IV. Examples And Exercises

I. Info Service

Overview

`info` is a service within dCache, which collects information about the respective cluster and provides it as an E**x**tensible **M**arkup **L**anguage (XML) document.

It is primarily intended for external use and currently not internally utilised by dCache itself.

The service is implemented by the `info` cell, which usually runs within the `infoDomain` domain.
`info` may run on any node of the cluster, but usually there should not be more than one instance of it.

The information is aggregated with a polling mechanism, where the `info` cell reads regularly data from other dCache components.

Those components are organised in “classes”, for example global data, pools selection configuration, pools and space reservations.
In order to prevent delays there is a minimum gap between querying objects of the same class and another minimum gap between querying different classes.

Overview

The following technical notes should be given:

- The effective polling frequency depends on the specific cluster.
- The classes correspond to the "main-elements" (for example `summary`, `domains`, `unitgroups`, `doors`, `pools`, `reservations`, et cetera) of the XML document, but they are not equal.
- Currently, the pool information is read from the `PoolManager` and not from the pools themselves.
  This may lead to discrepancies.

The `info` service provides also several commands within dCache's administration interface.

Previous Information Systems And Future Developments

dCache also contains the now deprecated `infoProvider` service, which was further used by an "infoProvider-based-infoProvider" to generate GLUE LDIF data.

It is only capable of providing very limited information and has been replaced by the `info` service (and additionally the "info-based-infoProvider" which makes use of it).

Future developments may include:

- Including more information.
- Implementing a push mechanism (at least for some parts) in order to get a better latency.

Data Interfaces

The `info` service's XML document is provided via the following interfaces:

- Raw TCP
  The `info` cell itself exports the complete XML document on the loopback network interface of the node where it runs via raw TCP on port `22112`.
  This is not accessible from remote hosts.
- HTTP
  If the `httpd` service is available it exports the complete XML document on all network interfaces of the node where it runs via HTTP on the configured port under the path "`/info/`".
  The HTTP port is specified by the `httpdPort`-configuration-parameter in *`dCache-home`*`/config/httpdoorSetup` on the node where it runs. The default value for this is "`2288`".

  It is also possible to request only a section of the XML document by appending the name of the respective elements (for example `summary`, `domains`, `unitgroups`, `doors`, `pools`, `reservations`, et cetera or even `pools/`*`pool-name`*, `pools/`*`pool-name`*`/queues`, et cetera) to the path.

Structure And Semantics Of The XML Document

The XML document has a tree-structure with exactly one "root-element", named "`dCache`", that contains several "main-elements" (for example `summary`, `domains`, `unitgroups`, `doors`, `pools`, `reservations`, et cetera).
The XML namespace, which is set in the "root-element", contains the version of the document format and is currently defined to be "`http://www.dcache.org/2008/01/Info`".

The structure of each "main-element" is specific to the information it holds.

The following list gives an overview of the semantics of the current "main-elements":

- `summary`
  Summary information (for example storage spaces, et cetera) about the overall storage, link groups and space reservations.
- `doors`
  Information (for example network values, protocol type and version, et cetera) about every door.

Structure And Semantics Of The XML Document

- `linkgroups`
  Information (for example ID, name, allowed access latencies and retention policies, storage spaces, authorisations, existing reservations, et cetera) about every link group.
- `unitgroups`
  Information (for example included units, referencing links, et cetera) about every unit group.
- `domains`
  Information (for example included cells and information about them, routing data, et cetera) about every domain.
- `links`
  Information (for example name, preferences, used units and unit groups, used pools and pool groups, storage spaces, et cetera) about every link.
- `nas` (“Normalised Access Space”)
  Information (sets of pools that are accessible by exactly the same storage units) that is for example used for accounting and “storage authorisation discovery”.

Structure And Semantics Of The XML Document

- `pools`
  Information (for example name, status, queue metrics, storage spaces, referencing pool groups, et cetera) about every pool.
- `units`
  Information (for example name, type, referencing unit groups, et cetera) about every unit.
- `reservations`
  Information (for example ID, description, status, access latency, retention policy, storage spaces, authorisations, link group, et cetera) about every space reservation.
- `poolgroups`
  Information (for example name, referencing links, used pools, storage spaces, et cetera) about every pool group.

II. GLUE

Overview

The GLUE schema is an abstract modelling for grid entities and resources.

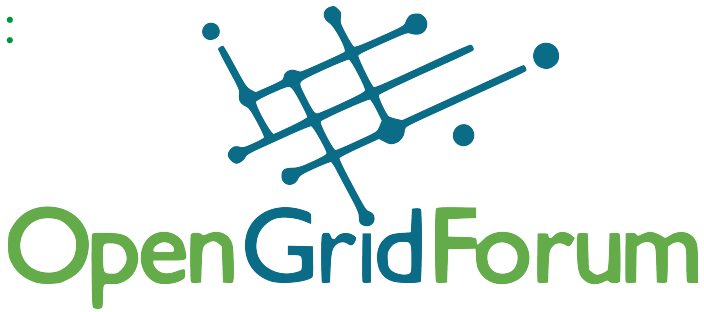
It is used within **G**rid **I**nformation **S**ervices (GIS) for several purposes including (and most important) the discovery and selection of grid services.

GLUE also defines concrete representations for this data in several formats, for example XML, LDAP and SQL.

GLUE is maintained by the Open Grid Forum's GLUE Working Group.

Examples of programms that use GLUE data include:

- Service Availability Monitoring
- File Transfer Service
- LCG Utilities
- miscellaneous ATLAS software



History And Versions

GLUE, which was originally an acronym for "**G**rid **L**aboratory **U**niform **E**nvironment", started as collaborative effort between the EU's DataTAG (**Data** **T**rans**a**tlantic **G**rid) and the USA's iVDGL (**I**nternational **V**irtual **D**ata **G**rid **L**aboratory) projects.

Currently, there are two major version branches:

- 1.X (includes the versions 1.0, 1.1, 1.2 and 1.3)
  The original schema, with several additions and corrections, which is still widespread among many grids including the LCG.
  The most recent version 1.3 defines only a representation-format for LDAP.
- 2.X (includes the version 2.0)
  A major redesign of GLUE under the aegis of the GLUE WG, that is incompatible to the previous versions.
  The current version 2.0 defines representation-formats for XML, SQL and LDAP.

This chapter only handles version 1.3.

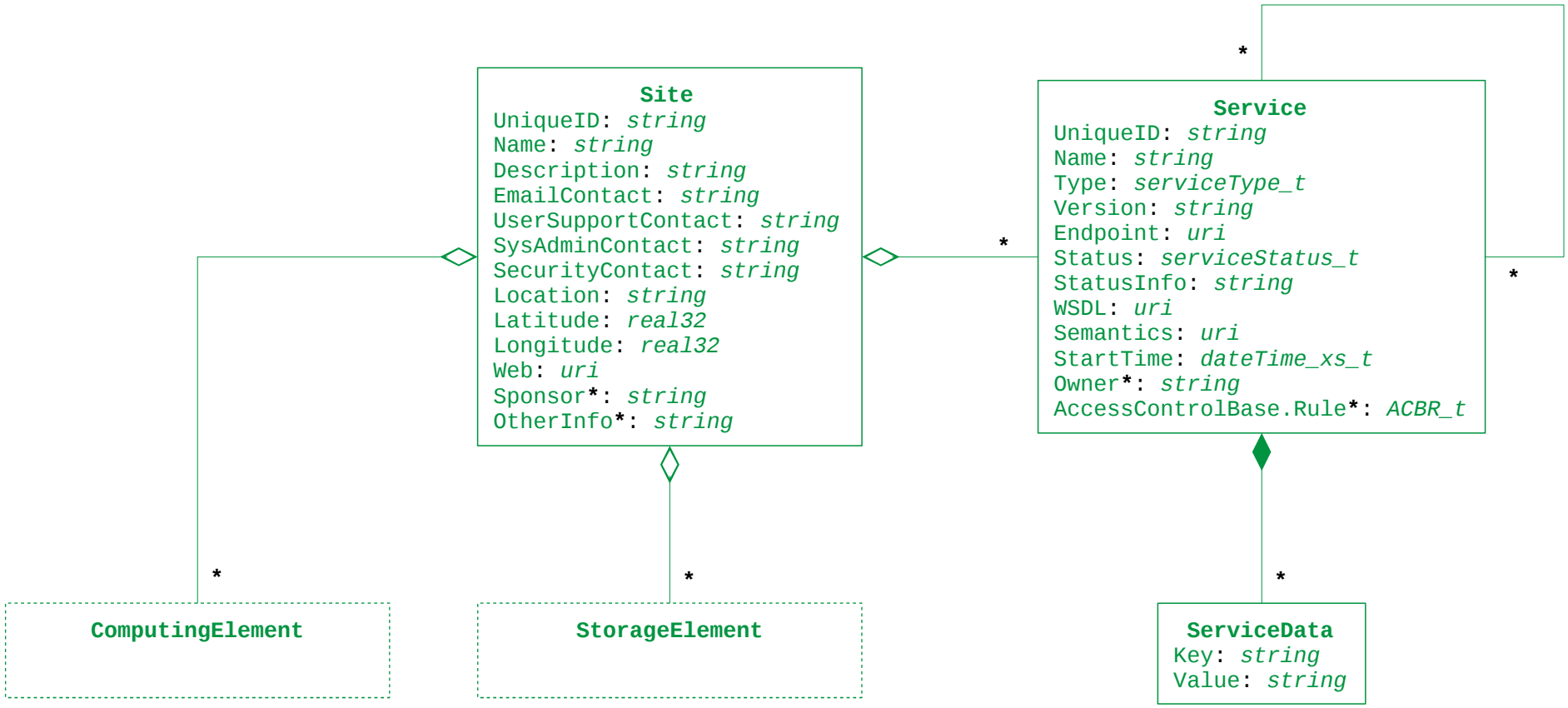
Standards Literature

The following lists links to standards literature that might be of interest:

- GLUE 1.0 (historical)
  http://www.cnaf.infn.it/~andreozzi/datatag/glue/CE/glueCE.htm
  http://www.cnaf.infn.it/~andreozzi/datatag/glue/SE/glueSE.htm
  http://www.cnaf.infn.it/~andreozzi/datatag/glue/CESE/glueCESE.htm
- GLUE 1.1 (historical)
  http://www.cnaf.infn.it/~andreozzi/datatag/glue/v11/CE/index.htm
  http://www.cnaf.infn.it/~andreozzi/datatag/glue/v11/SE/index.htm
  http://www.cnaf.infn.it/~andreozzi/datatag/glue/v11/CESE/index.htm
  http://www.cnaf.infn.it/~andreozzi/datatag/glue/v11/NE/index.htm
- GLUE 1.2 (historical)
  http://glueschema.forge.cnaf.infn.it/Spec/V12
- GLUE 1.3
  http://glueschema.forge.cnaf.infn.it/Spec/V13
- GLUE 2.0
  http://forge.gridforum.org/sf/go/projects.glue-wg/docman
- GLUE within LCG
  https://twiki.cern.ch/twiki/pub/LCG/WLCGCommonComputingReadinessChallenges/WLCG_GlueSchemaUsage-1.8.pdf

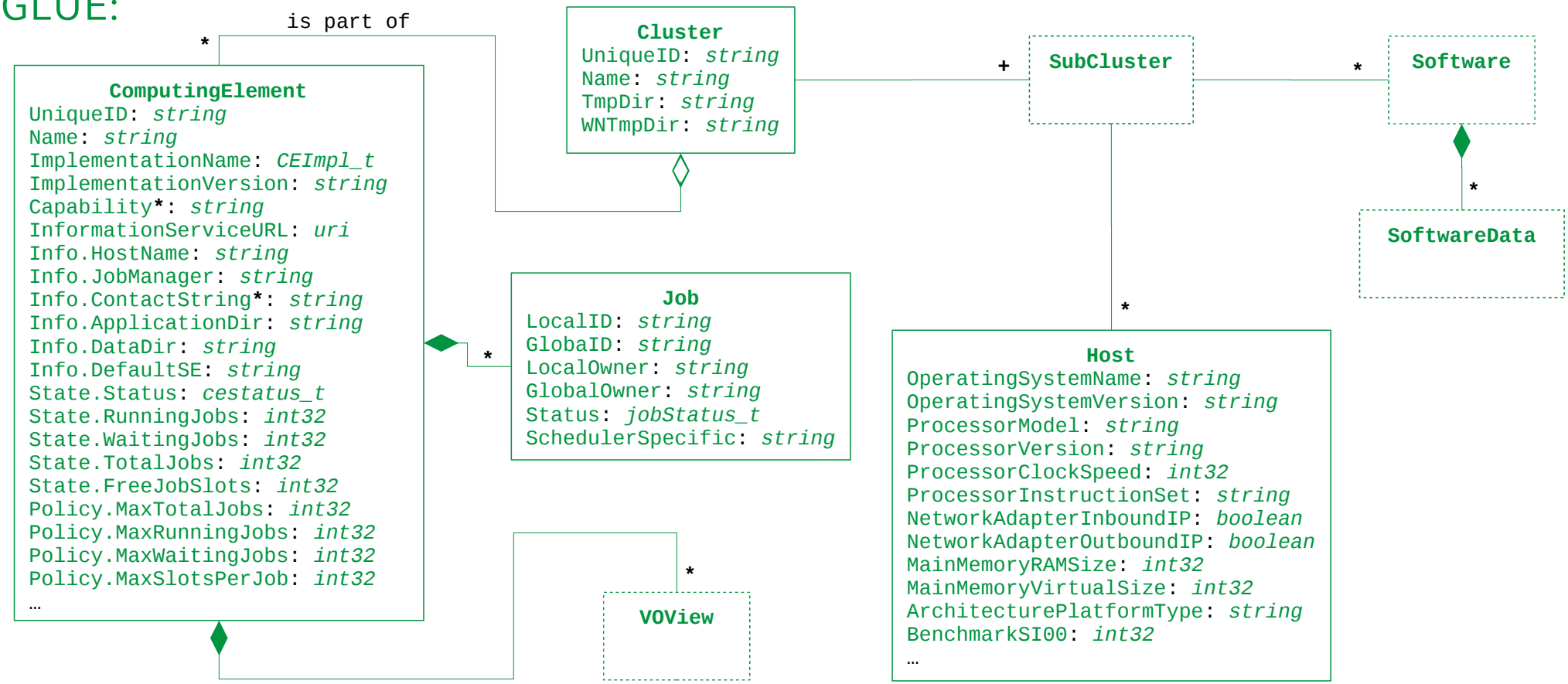
Structure – Core Entities

The following UML-diagram shows the structure of core entities within GLUE:



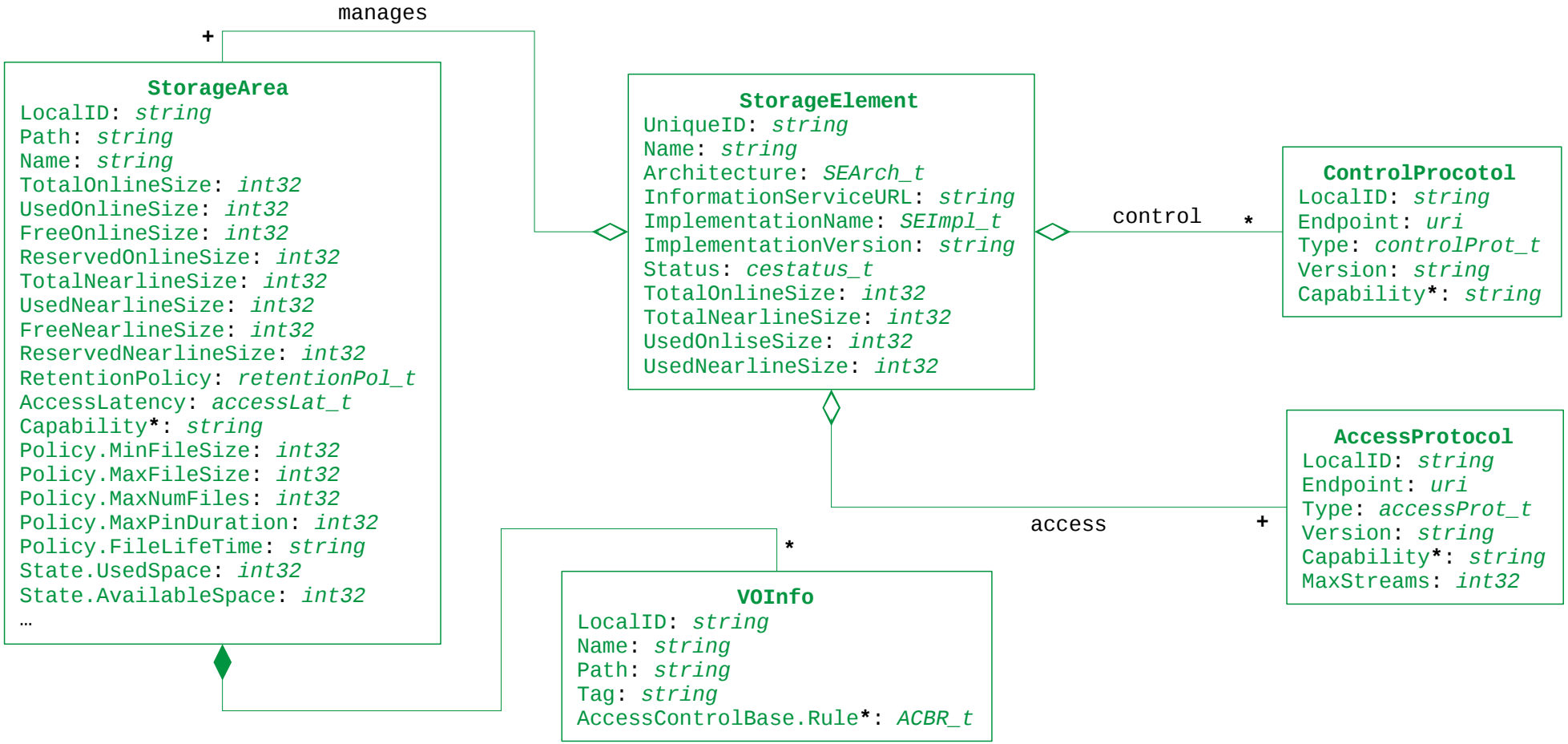
Structure – Computing Resources

The following UML-diagram shows the structure of computing resources within GLUE:



Structure – Storage Resources

The following UML-diagram shows the structure of storage resources within GLUE:



Generating GLUE Storage Resource Data Within dCache

dCache provides means to automatically generate GLUE version 1.3 storage resource data in the LDAP representation-format.

The "info-based-infoProvider" (which must not be confused with the deprecated `infoProvider` service) uses dynamic data from the `info` service as well as some static information in order to generate the LDIF (**L**DAP **D**ata **I**nterchange **F**ormat) output, which can for example be feed into a BDII-server to be subsequently used within a GIS.

Technically, a configuration file (per default *dCache-home*`/etc/glue-1.3.xml`), which also contains the static information mentioned above, is used by Xylophone (a set of XSLT stylesheets for converting XML data into LDIF) in conjunction with an XSLT processor (currently either Saxon or xsltproc) for the generation process.

The script *dCache-home*`/libexec/infoProvider/info-based-infoProvider.sh` executes this process and writes the resulting LDIF data to the standard output.

It must be able to query the `info` service via HTTP.

Configuring The "info-based-infoProvider"

The "info-based-infoProvider" itself is configured via the following parameters in *dCache-home*`/config/dCacheSetup`:

- `httpHost`
  Specifies the domain name of the node where the `info` service runs.
- `httpPort`
  Specifies the configured port on which the `info` service can be queried via HTTP.
- `xylophoneConfigurationDir`
  Specifies the directory of the configuration file for Xylophone.
- `xylophoneConfigurationFile`
  Specifies the name of the configuration file for Xylophone.
- `xsltProcessor`
  Specifies the XSLT processor to be used. Possible values are `saxon` for Saxon (the recommended default) and `xsltproc` for xsltproc.

Configuring The Generation Process And Static Information

The configuration file for Xylophone controls the generation process itself and specifies additionally required static information, that cannot be retrieved from the `info` service.

A template for this configuration file is located at *`dCache-home`*`/etc/glue-1.3.xml.template`.

It contains detailed instructions on its semantics and about how to set it up.

The template may change between different versions of dCache, including major modifications.

Therefore it is important to merge such changes into the actually used configuration file!

III. Checksum Module

Overview

The Checksum Module implements data integrity functionalities within dCache.

It provides automatic verification services on:

- incoming transfers from clients with submitted checksums ("`ontransfer`"),
- other writing, for example if the client did not submit checksums or in case of pool-to-pool-transfers ("`onwrite`") and
- restoring from HSM ("`onrestore`").

Not yet implemented are automatic verification services on:

- reading ("`onread`") and
- periodical checks.

Verification is also possible manually via:

- dCache's administration interface.

Overview

Technically, the Checksum Module is part of each pool and the file checksums are stored as meta-data within the file hierarchy provider (for example Chimera).

Each file can have multiple checksums from different algorithms stored.

Currently, the hash algorithms Adler-32 (which is the default), MD5 and MD4 are supported for storing.

For verification however, dCache always uses Adler-32 and ignores others if there are any.

Checksumming On Incoming Transfers

When a client writes data to the cluster the following happens regarding to checksumming:

1. Calculating the checksum at client-side (the so called “client checksum”).
   - DCAP/gsiDCAP
     The checksum may be calculated before the transfers starts and submitted with the “open-operation” or it may be calculated on-the-fly and submitted with the “close-operation”.
     Most clients use the later mode per default.
   - GridFTP
     In principle, checksumming is not supported by FTP.
     However, some clients can use the `SITE` (“site parameters”) command to specify a checksum, which is then calculated and submitted before the transfer starts.
2. Calculating the checksum at server-side (the so called “transfer checksum”).
   The server may calculate the checksum of the incoming data on-the-fly.

Checksumming On Incoming Transfers

3. Comparing the transfer checksum with the client checksum at server-side.
   If a client checksum was submitted and if a transfer checksum was calculated, compare them and result in an error if they differ.
4. If necessary, calculating the checksum at server-side (the so called “server file checksum”).
   If the server did not calculate the checksum in step 2, for example because multiple streams were used, calculate a checksum of the data using the already and completely stored file.

Configuration

The Checksum Module is configured per pool using dCache's administration interface.
The following configuration commands are provided:

- `csm set checksumtype` *`algorithm`* (ignored)
  Sets the used algorithm used for checksums to *`algorithm`*.
  dCache ignores this internally and always uses Adler-32 for verification.
- `csm set policy [-`*`event`*`=(on|off)]`* `[`*`options`*`]`
  Sets the policy of the Checksum Module, namely on which events checksumming should occur (set to `on`) or not (set to `off`).
  The following events exist and may be given as value for *`event`*:
  - `ontransfer`
    Checksumming on incoming transfers from clients with submitted checksums.
  - `onwrite`
    Checksumming on other writing, for example if the client did not submit checksums or in case of pool-to-pool-transfers.

Configuration

- `onrestore`
  Checksumming on restoring from HSM.
- `onread`
  Checksumming on reading (before opening the file).
  This is not yet implemented and silently ignored.
- `frequently`
  Automatic periodical checksumming of all files on the pool.
  This is not yet implemented and silently ignored.

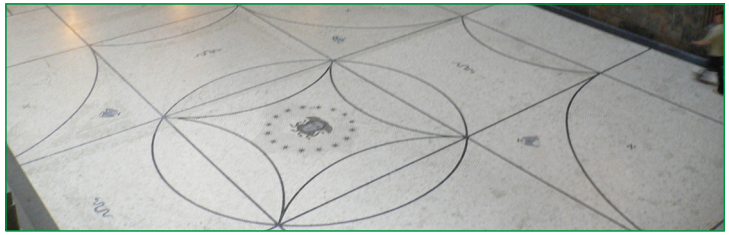
- `csm set policy -enforcecrc=(on|off)`
  Sets the policy of the Checksum Module, namely whether generation of checksums is enforced (set to `on`) or not (set to `off`).
- `csm info`
  Displays miscellaneous information, including the current policy and the status of any verification processes.

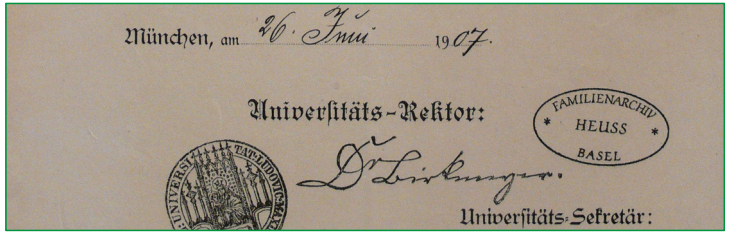
Manual Verification Of Files

Manually checking the integrity of files is possible via dCache's administration interface, where the following commands are provided:

- `csm check (*|`*`pnfs-id`*`)`
  Checks the file integrity of (if `*` was given) all files on the pool or a single file (if its PNFS-ID was given).
- `csm show errors`
  Displays any errors that were found.

IV. Examples And Exercises





Conventions And Assumptions

The following conventions are used:

- Lines starting with "`$`" are entered within a POSIX-`sh`-compatible shell.
- Lines starting with "`#`" are entered within a POSIX-`sh`-compatible shell, with the effective user-ID and group-ID being `0` ("root-rights").
- Lines starting with "`(`*`location`*`) >`" are entered within dCache's administration interface with *`location`* as the current location.
- Standard input is written black, standard output grey and standard error red.

The following assumptions are made:

- A basic knowledge on dCache, its administration interface and dCache-related clients exists.
- Chimera is used as file hierarchy provider.
- PostgreSQL is used as database management system.

1. Info Service

E1: Enabling The Info Service And Its Dependencies

The goal of this exercise is to enable the `info` service.

1. Enable the `httpd` service on one node of the cluster via one of the following ways:
   - If the `NODE_TYPE`-configuration-parameter in *dCache-home*`/etc/node_config` is set to `admin`, the `httpd` service is automatically started on that node.
   - Add "`httpd`" to the `SERVICES`-configuration-parameter in *dCache-home*`/etc/node_config`.
2. Enable the `info` service on one node of the cluster via one of the following ways:
   - If the `NODE_TYPE`-configuration-parameter in *dCache-home*`/etc/node_config` is set to `admin`, the `info` service is automatically started on that node.
   - Add "`info`" to the `SERVICES`-configuration-parameter in *dCache-home*`/etc/node_config`.
3. Start the two services via:
   ```
   # dCache-home/bin/dcache start http
   # dCache-home/bin/dcache start info
   ```
4. Check whether the two services are actually running via:
   ```
   $ dCache-home/bin/dcache status
   ```

E2: Read The XML Document Via Raw TCP

The goal of this exercise is to read the XML document via raw TCP.

The following must be done on the node where `info` service runs.

1. Read the complete XML document using netcat via:
   ```
   $ nc localhost 22112
   ```

E3: Read The XML Document Via HTTP

The goal of this exercise is to read the XML document via HTTP.

The following can be done on any host that is able to connect via HTTP to the node where the `httpd` service runs on the configured port.

It is assumed, that the node where the `httpd` service runs can be reached via the domain name "`httpd.dcache.example.org`". Further, the default value "`2288`" is assumed for the configured port.

1. Read the complete XML document using wget via:

```
$ wget --output-document=info0.xml http://localhost:2288/info/
$ wget --output-document=info1.xml http://httpd.dcache.example.org:2288/info/
```

2. Read single sections of the XML document using wget via:

```
$ wget --output-document=info2.xml http://localhost:2288/info/summary
$ wget --output-document=info3.xml http://localhost:2288/info/domains
$ wget --output-document=info5.xml http://localhost:2288/info/doors
$ wget --output-document=info6.xml http://localhost:2288/info/pools/
$ wget --output-document=info6.xml http://localhost:2288/info/pools/pool-name
$ wget --output-document=info6.xml http://localhost:2288/info/pools/pool-name/space
```

`pool-name` must be replaced with a valid pool name, for example "`pool_1`".

E4: Learning The Provided Information

The goal of this exercise is to learn the provided information.

1. Read through the complete XML document and try to understand the semantics of the different elements, attributes and values.

E5: The Info Cell Within dCache's Administration Interface

The goal of this exercise is to explore some of the commands provided by the `info` cell within dCache's administration interface.

1. Within dCache's administration interface, enter the `info` cell.
2. Try out the following commands:
   - `info [-a] [-l]`
     Prints general information about the `info` cell itself.
   - `state ls [path]`
     Prints the current information collected by the `info` service.
     Specifying a `path` allows a restriction of the output.
   - `dga (ls|(enable|disable|trigger) name)`
     This suite of commands allows to show (...`ls`), to enable (...`enable` *name*) or disable (...`disable` *name*) as well as to trigger (...`trigger` *name*) data gathering activities.

2. GLUE

E1: Setting Up glue-1.3.xml

The goal of this exercise is to set up the Xylophone configuration file `glue-1.3.xml`, which controls the generation process and specifies additionally required static information.

1. Merge the most recent version of the configuration file template *dCache-home*`/etc/glue-1.3.xml.template` into the actual configuration file *dCache-home*`/etc/glue-1.3.xml`.

   If the later does not yet exist, simply copy the template.
2. If a non-default location was used for the configuration file, set the parameters `xylophoneConfigurationDir` and `xylophoneConfigurationFile` in *dCache-home*`/config/dCacheSetup` accordingly.
3. Follow the instructions in the template file to configure all the required information.

   This step is heavily dependant on the specific site, particularly factors like general site data, supported VOs and information about storage areas.

E2: Checking The LDIF Output

The goal of this exercise is to check the LDIF output generated by the "info-based-infoProvider".

1. Execute the script *dCache-home*`/libexec/infoProvider/info-based-infoProvider.sh` and redirect its standard output into a temporary file via:
   ```
   $ dCache-home/libexec/infoProvider/info-based-infoProvider.sh > /tmp/GLUE.ldif
   ```
2. Check whether any errors or warnings have been written to standard error.
3. Read the generated LDIF output and try to understand its structure and semantics.

E3: Control Which Access Protocols Are Published

The goal of this exercise is to control which access protocols are published and which not.

The configuration file template *dCache-home*`/etc/glue-1.3.xml.template` is configured to not publish information about the access protocols DCAP and xrootd ("proofd").

This is achieved with stanzas like the following:

```
<suppress test="dcap">
   <lookup path="d:protocol/d:metric[@name='family']"/>
</suppress>
```

In order to enable publishing simply comment the respective stanza like the following:

```
<!--
<suppress test="dcap">
   <lookup path="d:protocol/d:metric[@name='family']"/>
</suppress>
-->
```

1. Try out to enable the publishing of DCAP and xrootd ("proofd") information.

E4: Decommissioning Of The Deprecated InfoProvider

The goal of this exercise is to decommission the deprecated InfoProvider, if it was used.

1. Stop the `infoProvider` service via:
   ```
   # dCache-home/bin/dcache stop infoProvider
   ```
2. Remove "`httpd`" from the `SERVICES`-configuration-parameter in *`dCache-home`*`/etc/node_config`.
3. If the **G**eneric **I**nformation **P**rovider (GIP) was used:
   1. Remove the following files:
      - `/opt/glite/etc/gip/ldif/lcg-info-static-SE.ldif`
      - `/opt/glite/etc/gip/ldif/lcg-info-static-dSE.ldif`
      - `/opt/glite/etc/gip/plugin/infoDynamicSE-plugin-dcache`

      Depending on GIP's version and configuration, its configuration directory might be located at a different path (for example `/opt/lcg/var/gip/`).
      In addition, the files might have different names or be located in different subdirectories.
   2. Other files that are now obsolete might have been used with GIP and must be deleted, too.

E4: Decommissioning Of The Deprecated InfoProvider

4. If the 3rd-party “Space Token InfoProvider” was used:
   1. Remove any symbolic links within GIP’s configuration directories to one of the following files of the “Space Token InfoProvider”:
      - `info_provider.py`
      - `token_info_provider.py`

      The links are typically found in `/opt/glite/etc/gip/provider/`.

E5: Publishing The "info-based-infoProvider"'s LDIF Output

The goal of this exercise is to publish the "info-based-infoProvider"'s LDIF output.

If the **G**eneric **I**nformation **P**rovider (GIP) is used:

1. Create a symbolic link to the script *dCache-home*`/libexec/infoProvider/info-based-infoProvider.sh` within the GIP's "provider-directory" for example via:
   ```
   # ln -s dCache-home/libexec/infoProvider/info-based-infoProvider.sh
   /opt/glite/etc/gip/provider/
   ```
   The "provider-directory" might be found at a different location (for example `/opt/lcg/var/gip/provider/`).

It is also possible to let a BDII-server directly query the "info-based-infoProvider" without using an intermediate GIP:

1. Add the "info-based-infoProvider" as a provider by adding the following line to the configuration file `bdii-update.conf`:
   ```
   ProviderA file://dCache-home/libexec/infoProvider/info-based-infoProvider.sh
   ```
   It might be necessary to change the provider-ID *A* to the next free letter.
   As always, *dCache-home* must be replaced with the absolute path to dCache,s for example `/opt/d-cache`.

E6: Checking The Validity Of The Published GLUE Data With GStat

The goal of this exercise is to check the validity of the GLUE data published by production sites with GStat.

The following assumes the usage of GStat version 2.0.

1. Open GStat which is available as webservice at http://gstat-prod.cern.ch/.
2. Select "Site Views", then the grid or organisation the desired site belongs to and then the site itself.
3. Now select a hostname in the frame "Information System Monitoring and Validation".
4. The frame "Site Resource Components" shows you several checks, for example "check-se", which checks the published storage resource data.
5. The frame "Viewing Testing Results or Statistics Graphs" shows the number of warnings, errors and infos that were found.
   Details are displayed when the item is expanded.

3. Checksum Module

E1: Preparations

The goal of this exercise is to make preparations for the following exercises.

This and the following exercises operate on one given pool, which is named *pool1_1* here. Adapt the name as required.

1. Verify that checksum calculation is enabled for at least the "onwrite"-event via:
   ```
   (pool1_1)> csm info
   ...
          on write : true
   ...
   ```
   If it is not, enable it via:
   ```
   (pool1_1)> csm set policy -onwrite=on
   ```
2. Some test files (at least one) will be required on *pool1_1*. Upload a few files to the dCache cluster, for example with dccp.

   If more than one pool is used, dCache may likely spread the files over them. This can however be prevented, for example by setting the other pools to read only.

E1: Preparations

3. Determine the "pool-directory" of *pool1_1* on the node where it runs via:

```
# dCache-home/bin/dcache pool ls
Pool    Domain      LFS      Size Free Path
...
pool1_1 pool1Domain precious 1000 500  /srv/dcache/pool1/
...
```

   Here, the "pool-directory" is "*/srv/dcache/pool1*".

4. Note, that a "pool-directory" contains about the following files and directories:

pool-directory/

- Contains all data and files the pool consists of.

`controls/`

Contains two "control-files" for each actual file in the "data-directory" with meta-data about it.

`data/`

Contains all the actual files from the pool, each named after its PNFS-ID.

`setup`

Contains all settings (for example size, queues, et cetera) of the pool.

E1: Preparations

5. dCache defines the following “checksum-types”:
   - 1: Adler-32
   - 2: MD5
   - 3: MD4

   They are used in several places including dCache’s administration interface and its databases.

E2: Reading The Stored Checksum Values Of Files

The goal of this exercise is to read the stored checksum values of files.

1. Select a file on *pool1_1* and determine its PNFS-ID.
   Here we assume the PNFS-pathname */pnfs/example.org/data/test-file* and the PNFS-ID *0000E316BE29EAA64AF3940E92E636129CD0*.
2. Read the stored checksums of the file directly from the `chimera` database via the following SQL-query:

   ```
   SELECT itype,isum FROM t_inodes_checksum WHERE
   ipnfsid='0000E316BE29EAA64AF3940E92E636129CD0';
   ```

   `itype` specifies the respective "checksum-type".

   When using the `psql`-client, this can be done for example via:

   ```
   # psql --username postgres --dbname chimera --command "SELECT itype,isum FROM
   t_inodes_checksum WHERE ipnfsid='0000E316BE29EAA64AF3940E92E636129CD0';"
   ```

3. Read the stored checksums of the file in dCache's administration interface via:

   ```
   (PnfsManager) > get file checksum 0000E316BE29EAA64AF3940E92E636129CD0 1
   (PnfsManager) > get file checksum 0000E316BE29EAA64AF3940E92E636129CD0 2
   (PnfsManager) > get file checksum 0000E316BE29EAA64AF3940E92E636129CD0 3
   ```

   The last argument specifies the "checksum-type" to be queried.

E3: Verifying The Integrity Of Files

The goal of this exercise is to verify the integrity of files.

1. Select a file on *pool1_1* and determine its PNFS-ID.
   Here we assume the PNFS-pathname */pnfs/example.org/data/test-file* and the PNFS-ID *0000E316BE29EAA64AF3940E92E636129CD0*.
2. Trigger the checksum verification via:
   ```
   (pool1_1) > csm check 0000E316BE29EAA64AF3940E92E636129CD0
   ```
   Only the stored Adler-32 checksums are validated.
3. Wait until the verification process has already finished, which can be checked via:
   ```
   (pool1_1) > csm info
   ```
   If it has, the `SingeScan`-status is `Idle`, if not it is `Active`.
4. Display any errors that were found via:
   ```
   (pool1_1) > csm show errors
   ```
5. Optionally, verify all files on the pool via:
   ```
   (pool1_1) > csm check *
   ```
   Repeat steps 3 and 4.
   The process status of the full scan is given by `FullScan`-status.

E4: Simulating Corrupted Files

The goal of this exercise is to simulate corrupted files and check whether verifcation errors occur.

1. Select a file on *pool1_1* and determine its PNFS-ID.
   Here we assume the PNFS-pathname */pnfs/example.org/data/test-file* and the PNFS-ID *0000E316BE29EAA64AF3940E92E636129CD0*.

   The file is physically located on the node that runs *pool1_1* at the pathname *pool-directory*/data/*0000E316BE29EAA64AF3940E92E636129CD0*, where *pool-directory* is the "pool-directory" as described in exercise 1.
2. Corrupt the file, for example by overwriting it with a new file (optionally, of the same size) filled with random data via:
   ```
   # pathname="pool-directory/data/0000E316BE29EAA64AF3940E92E636129CD0"
   # size="$(stat --format=%s "${pathname}")"
   # dd if=/dev/urandom of="${pathname}" bs=1 count="${size}"
   ```
3. Repeat steps 2 to 4 from exercise 3, for the file selected in step 1.

Acknowledgements

The following people contributed to create this presentation (given in alphabetical order):

- Fuhrmann, Patrick
- Millar, Paul
- Mkrtchyan, Tigran
- Mol, Xavier

Finis coronat opus.