LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

3rd dCache Workshop
Rheinisch-Westfälische Technische Hochschule Aachen
Aachen April 8, 2009

Access Control within dCache

gPlazma and ACLs

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

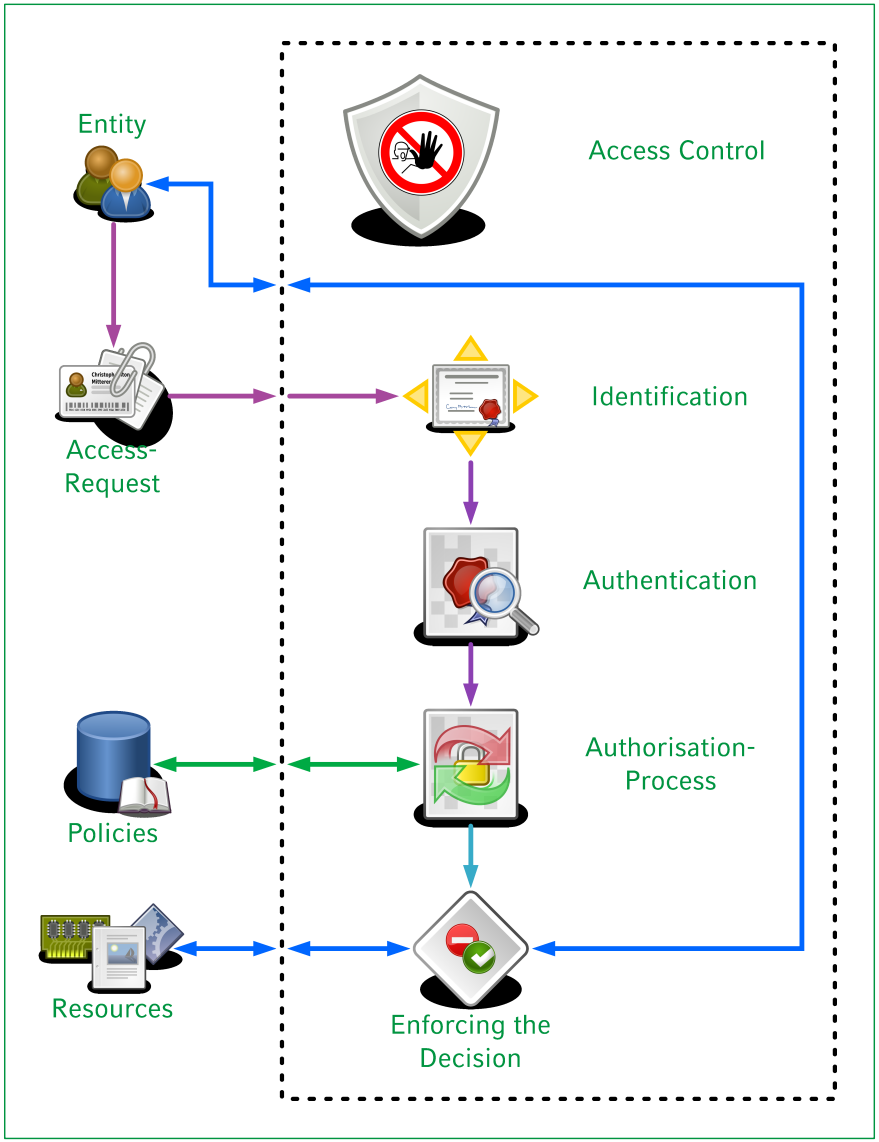
Contents

Contents

I. Introduction

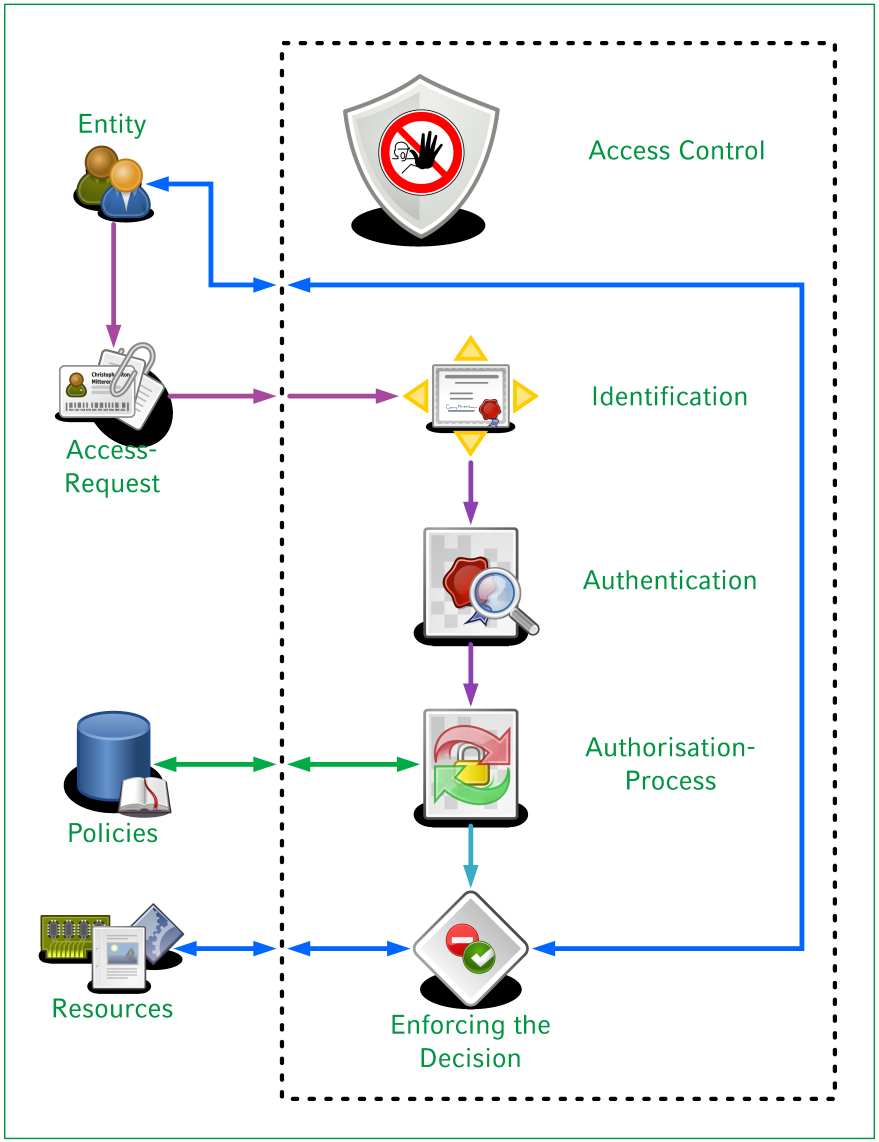
General Concepts

- *Entities* and *Resources*
  Entities (for example persons, groups or processes) are making access requests, using one of their identities, in order to get access to resources (for example files, services provided by hardware or functionality provided by software).
- *Identities*
  An identity is a set of attributes describing its owner (an entity).
  Identities are created by identity-providers, which validate and certify the correctness of the attributes.
- *Policies*
  Policies contain the rules that describe how resources might be accessed.



General Concepts

- *Access-Request*
  A client's request to access resources.
- *Identification*
  The act of presenting an identity.
- *Authentication*
  The process of verifying the integrity and authenticity of an identity, using the credentials provided by the identity-provider.
- *Authorisation-Process*
  The process of finding a decision to the access-request, using the data from the identity, the policies and the access-request itself.
- *Enforcing the Decision*
  Granting or denying the access.



Motivation

General reasons for access control are:

- Integrity
- Confidentiality
- Availability

… of the protected resources, which are primarily files in the case of storage systems.

From the perspective of LCG Storage Elements this looks like the following:

- Confidentiality is less important, as most data is public anyway.
- Availability is mainly reached by redundancy and automatic replication services, but not via means of access control.
- Integrity is definitely the most important aim of access control, as different classes of data must be secured from being modified or even deleted accidentally.

  A typical example is, that normal users should not be able to write in “production-storage-areas”.

Access Control Systems in dCache

dCache contains several access control systems, used with different types of protocols and providing a varying set of features:

- dcache.kpwd (dcap, GridFTP, SRM)
  This is a legacy system that was used before the implementation of gPlazma and is still used by dcap-door-cells. It should not be confused with gPlazma's "kpwd"-plug-in.

  Using this system with GridFTP- or SRM-door-cells is still possible [1], but strongly discouraged.
- xrootd-ALICE (xrootd)
  This system implements the so called "ALICE security model" and is only used by xrootd-door-cells.
- gPlazma + door-cell (GSIdcap, GridFTP, SRM)
  The GSIdcap-, GridFTP- and SRM-door-cells implement their own (very similar) access control systems by using the services provided by gPlazma.

This presentation focuses on gPlazma.

II. gPlazma – Overview and Configuration

General Information

gPlazma (**G**rid-Aware **Pl**uggable **A**uthori**z**ation **Ma**nagement) is a part of dCache, providing services for access control, which are used by door-cells in order to implement their access control system.

It should be noted, that usually not every type of door-cell makes use of all the features provided by gPlazma.

gPlazma was introduced with dCache version 1.7 and is currently used by GSIdcap-, GridFTP- and SRM-door-cells (gPlazma-support for xrootd-door-cells is planned).

In order to serve different needs, gPlazma utilises plug-ins as back-end for its tasks and services.
Currently, the following plug-ins are provided:

- kpwd
- grid-mapfile
- gplazmalite-vorole-mapping
- saml-vo-mapping
- xacml-vo-mapping

Requirements and Constraints

gPlazma has a number of requirements and constraints:

- Of course, the specific door-cell must support gPlazma, which is currently not the case with dcap- and xrootd-door-cells.
- There can be only one gPlazma-cell per dCache-cluster.
- gPlazma requires the CA- and VOMS-root-certificates, that it should use, to be present in `/etc/grid-security/certificates/` and `/etc/grid-security/vomsdir` respectively.
- In some cases, gPlazma requires X.509-host-certificates to be present in `/etc/grid-security/`.
  TODO
- The configuration for gPlazma and its used plug-ins must be present on any host that either invokes gPlazma as cell or directly as module.
- Multiple DNs per client-certificate are currently generally not supported (but multiple roles are).
- The gPlazma-cell must be restarted in order to notice changes to its configuration.

Enabling the gPlazma-cell and General Configuration

In order to enable the gPlazma-cell for a dCache-cluster the following must be set in *dCache-home*`/etc/node_config` on one node:

```
gPlazmaService=yes
```

Remember that there may be not more than one gPlazma-cell per dCache-cluster.

If gPlazma is neither used as cell, nor as module, GridFTP- and SRM-door-cells will fall back to the legacy access control system, “dcache.kpwd” (whose configuration must then be present on any host that makes use of it).

General configuration-parameters in *dCache-home*`/config/gPlazmaSetup` are:

- `gplazmaPolicy`
  Specifies the pathname of the “gPlazma-policy-configuration-file”. [2]
- `gPlazmaNumberOfSimutaneousRequests`
  Specifies the number of concurrent requests to gPlazma.
- `gPlazmaRequestTimeout`
  Specifies the number of seconds, in which a request must be answered by gPlazma before being denied.

Invoking gPlazma as Cell or as Module

There are two ways, how a door-cell can invoke gPlazma:

- as cell
  The door-cell communicates with the gPlazma-cell, which can be on any node of the dCache-cluster, using “dCache-cell-messaging”.
- as module
  The door-cell invokes the gPlazma-code locally.

  Note that in order for this being possible, all necessary configuration must be present on the nodes where gPlazma is invoked as module.
  Additionally, some of the general configuration-options might not work, when gPlazma is invoked as module.

Invoking gPlazma as Cell or as Module

For door-cells, supporting both invocation-ways of gPlazma, this can be configured on a per node-basis via the following parameters in the configuration-file of the respective door-cell:

- `useGPlazmaAuthorizationCell`
  Set to `true` if gPlazma should be invoked as cell, or to `false` otherwise.
- `useGPlazmaAuthorizationModule`
  Set to `true` if gPlazma should be invoked as module, or to `false` otherwise.

If both are set to `true`, the cell is queried first and if this fails (in case of errors) gPlazma is invoked as module. If both are set to `false`, gPlazma will not be used and dCache falls back to its legacy access control system.

Note that configuration- and policy-data might differ between the “cell-node” and the “module-nodes”, which may lead to conflicts.

Currently, this is supported by GSIdcap-, GridFTP-, and SRM-door-cells.

Selecting Plug-ins and specifying their Priorities

gPlazma's plug-ins are mainly configured via the "gPlazma-policy-configuration-file" (per default found at *dCache-home*`/config/dcachesrm-gplazma.policy`).

Plug-ins are enabled or disabled via the following configuration-parameters:

- `kpwd`, `grid-mapfile`, `gplazmalite-vorole-mapping`, `saml-vo-mapping` and `xacml-vo-mapping`
  Set to `ON` if the specific plug-in should be enabled, or `OFF` otherwise.

The order in which gPlazma tries the plug-ins until any mappings are found can be set via the following configuration-parameters:

- `kpwd-priority`, `grid-mapfile-priority`, `gplazmalite-vorole-mapping-priority`, `saml-vo-mapping-priority` and `xacml-vo-mapping-priority`
  Set to natural numbers starting with `1`, where a smaller number means a higher priority.

“kpwd”-Plug-in

This plug-in allows the usage of “dcache.kpwd-style”-files, as known from dCache’s legacy access control system, within gPlazma.

It should be noted, that this plug-in does not support some modern features, for example VO-attributes attached to certificates or secondary groups.

The plug-in is configured via the following parameters in the “gPlazma-policy-configuration-file”:

- `kpwdPath`
  Specifies the pathname of the “dcache.kpwd-style”-file to be used by the plug-in.
  This must not be confused with the `kpwdFile` configuration-parameter that is used[3] by dCache’s legacy access control system.

The syntax and semantics of “dcache.kpwd-style”-files are explained in *dCache-home*`/etc/dcache.kpwd.template`.

“grid-mapfile”-Plug-in

This plug-in allows the usage of “grid-mapfile-style”-files, as known from the Globus Toolkit, within gPlazma.

It should be noted, that this plug-in does not support some modern features, for example VO-attributes attached to certificates.

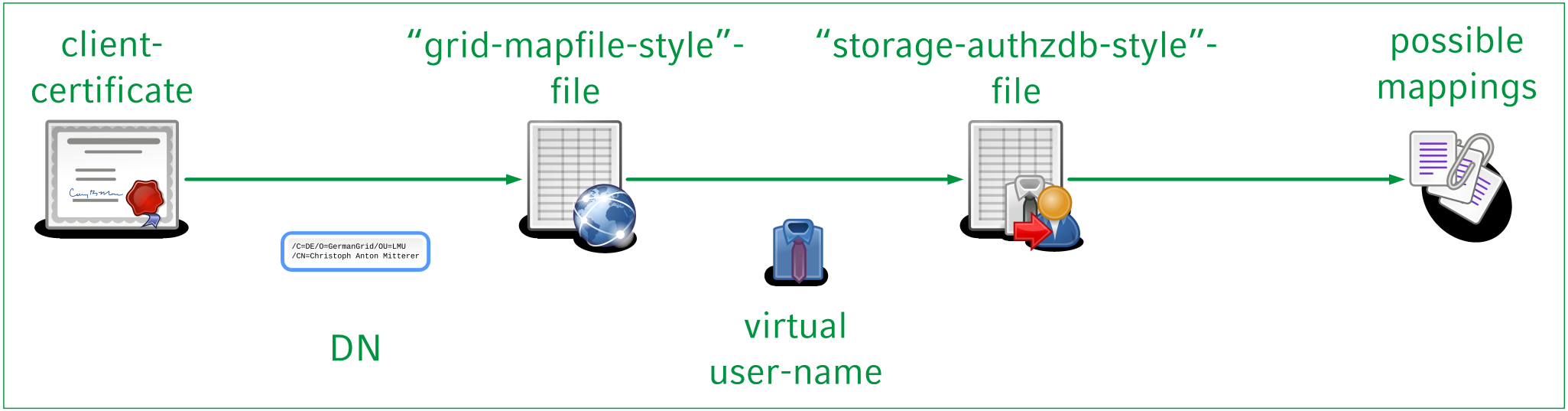
The plug-in is configured via the following parameters in the “gPlazma-policy-configuration-file”:

- `gridMapFilePath`
  Specifies the pathname of the “grid-mapfile-style”-file to be used by the plug-in.
- `storageAuthzPath`
  Specifies the pathname of the “storage-authzdb-style”-file to be used by the plug-in.

“grid-mapfile”-Plug-in

The following two-stage mapping-mechanism is applied by the plug-in:

1. Using the “grid-mapfile-style”-file, the client-certificate’s DN is mapped to a virtual user-name, which is not to be confused with an actual UNIX user-name.
2. Using the “storage-authzdb-style”-file, this virtual user-name is then mapped to the actual UNIX user-ID [4] and group-IDs [4] (as they are also used by dCache’s file-hierarchy-provider, which is either Chimera or the legacy PNFS) as well as some other information.



“grid-mapfile-Style”-File Syntax and Semantics

Each line specifies a mapping from a client-certificate’s DN to exactly one [5] virtual user-name via the following syntax:
`"distinguished_name" virtual_user-name`

distinguished_name must always be quoted using “ "”.
Lines not starting with “ "” are currently ignored by dCache.
If the same DN occurs in multiple lines then only the mapping from the last one is used.

Examples:

- `"/C=DE/O=GermanGrid/OU=LMU/CN=Christoph Anton Mitterer" atlas`
- `"/C=DE/O=GridGermany/OU=Leibniz-Rechenzentrum/ CN=Christoph Anton Mitterer" dgrid`

“grid-mapfile-Style”-File Syntax and Semantics

More information about the syntax and semantics of “grid-mapfile-style”-files can be found in the documentation to the Globus Toolkit.

In addition, the Globus Toolkit provides the `grid-mapfile-add-entry`, `grid-mapfile-delete-entry` and `grid-mapfile-check-consistency` programs to maintain “grid-mapfile-style”-files.

“gplazmalite-vorole-mapping”-Plug-in

This plug-in allows the usage of “grid-vorolemap-style”-files, which are similar to the “grid-mapfile-style”-files but provide support for virtual organisations and their attributes [6], like `Role` and `Capability`.

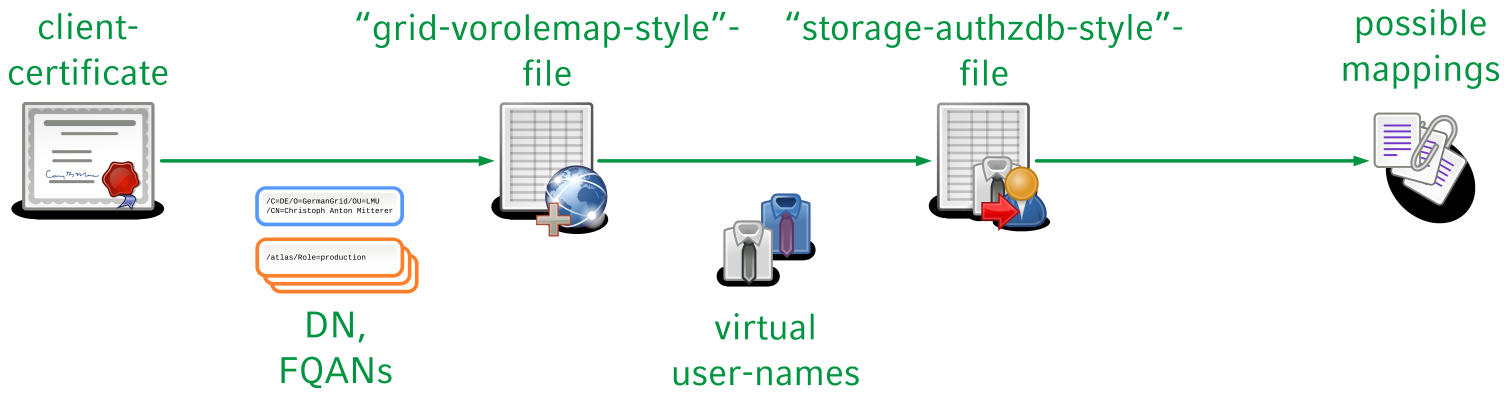
The plug-in is configured via the following parameters in the “gPlazma-policy-configuration-file”:

- `gridVoRolemapPath`
  Specifies the pathname of the “grid-vorolemap-style”-file to be used by the plug-in.
- `gridVoRoleStorageAuthzPath`
  Specifies the pathname of the “storage-authzdb-style”-file to be used by the plug-in.

“gplazmalite-vorole-mapping”-Plug-in

The following two-stage mapping-mechanism is applied by the plug-in:

1. Using the “grid-vorolemap-style”-file, the client-certificate’s DN and FQANs are mapped to some virtual user-names, which are not to be confused with actual UNIX user-names.
2. Using the “storage-authzdb-style”-file, these virtual user-names are then mapped to the actual UNIX user-ID(s) [4] and group-IDs [4] (as they are also used by dCache’s file-hierarchy-provider, which is either Chimera or the legacy PNFS) as well as some other information.



“grid-vorolemap-Style”-File Syntax and Semantics

Each line specifies a mapping from a client-certificate’s DN and FQAN to exactly one virtual user-name via the following syntax:

`"`*`distinguished_name`*`" ["`*`fqan`*`"] `*`virtual_user-name`*

`distinguished_name` and *`fqan`* must always be quoted using “ `""`[7].
Lines not starting with “ `"`” are currently ignored by dCache.
If the same DN occurs in multiple lines with the same FQAN then only the mapping from the last one is used. However, the same DN can be used multiple times with different FQANs and thus map to different virtual user-names.
If *`fqan`* is empty or `"`*`fqan`*`"` not specified at all, only client-certificates with an empty or no FQAN will match.

Examples:

- `"/C=DE/O=GermanGrid/OU=LMU/CN=Christoph Anton Mitterer" "/atlas" atlas001`
- `"/C=DE/O=GermanGrid/OU=LMU/CN=Christoph Anton Mitterer" "/atlas/Role=production" prdatl01`

“grid-vorolemap-Style”-File Syntax and Semantics

distinguished_name can also be set to “ *”, which serves as a regular expression matching any character sequence (“wildcard character”).
This is especially useful when mapping whole VOs.

Examples:

- `"*" "/atlas" atlas001`
- `"*" "/atlas/Role=production" prdatl01`

It is important to note, that when any mapping is found via an “explicit-DN-match”, all mappings that would arise from a “wildcard-match” are ignored. This applies for “disabling entries”, too.

“Disabling entries” (also called “revocation entries”) can made by using “ -” as virtual user-name.
This is a valid mapping and thus no “further” plug-ins will be tried. It should be noted however, that “previous” plug-ins are still tried. (See priority of plug-ins.)

“grid-vorolemap-Style”-File – Example Mappings

- “single wildcard-match”
  Certificate-DN: `/C=DE/O=GermanGrid/OU=LMU/CN=Christoph Anton Mitterer`
  Certificate-FQANs: `/atlas`
  “grid-vorolemap-style”-file-contents:

```
"*" "/atlas" atlas001
"*" "/atlas/de" atlas002
```

  Resulting mappings: `atlas001`
- “multiple wildcard-matches”
  Certificate-DN: `/C=DE/O=GermanGrid/OU=LMU/CN=Christoph Anton Mitterer`
  Certificate-FQANs: `/atlas`, `/atlas/de`, `/atlas/Role=production`
  “grid-vorolemap-style”-file-contents:

```
"*" "/atlas" atlas001
"*" "/atlas/de" atlas002
"*" "/atlas/Role=production" prdatl01
```

  Resulting mappings: `atlas001`, `atlas002`, `prdatl01`

“grid-vorolemap-Style”-File – Example Mappings

- “overriding explicit-DN-match”
  Certificate-DN: `/C=DE/O=GermanGrid/OU=LMU/CN=Christoph Anton Mitterer`
  Certificate-FQANs: `/atlas`
  “grid-vorolemap-style”-file-contents:
  (the order of these lines does not matter)

```
"*" "/atlas" atlas001
"/C=DE/O=GermanGrid/OU=LMU/CN=Christoph Anton Mitterer" "/atlas" ops
```

  Resulting mappings: `ops`
- “disabling entry”
  Certificate-DN: `/C=DE/O=GermanGrid/OU=LMU/CN=Christoph Anton Mitterer`
  Certificate-FQANs: `/atlas`, `/atlas/de`, `/atlas/Role=production`
  “grid-vorolemap-style”-file-contents:
  (the order of these lines does not matter)

```
"*" "/atlas" atlas001
"/C=DE/O=GermanGrid/OU=LMU/CN=Christoph Anton Mitterer" "/atlas" -
```

  Resulting mappings: `-` (special “disabling” mapping)

“saml-vo-mapping”-Plug-in

This plug-in allows the usage of the **S**ecurity **A**ssertion **M**arkup **L**anguage (SAML) within gPlazma, which is typically done, when a GUMS-server (Grid User Management System) should be used for providing mapping-services.

The plug-in is configured via the following parameters in the “gPlazma-policy-configuration-file”:

- `mappingServiceUrl`
  Specifies the URL of the mapping-service (typically the GUMS-server).
- `saml-vo-mapping-cache-lifetime`
  Specifies the caching lifetime of queried mappings in seconds. If set to `0` caching is disabled.
- `gridVoRoleStorageAuthzPath`[8]
  Specifies the pathname of the “storage-authzdb-style”-file to be used by the plug-in, which is only required when using the so called “GUMSAuthorization-ServicePort”, but not when using the “StorageAuthorizationServicePort”.

“saml-vo-mapping”-Plug-in

The following describes the basic procedure of using a GUMS-server:

1. If the mapping-result is not cached, the plug-in contacts the GUMS-server in a secure way (thus X.509 host-certificates an CA-root-certificates are required) [9], providing it with the whole certificate chain of the specific access-request.
2. The GUMS-server does the necessary work to determine the desired mapping and returns the result to the plug-in:
   a. If the “GUMSAuthorizationServicePort” was used, virtual user-names will be returned, which have to be resolved by the plug-in using its “storage-authzdb-style”-file.
   b. If the “StorageAuthorizationServicePort” was used, the actual UNIX user-ID and group-IDs (as they are also used by dCache’s file-hierarchy-provider, which is either Chimera or the legacy PNFS) as well as some other information will be returned.

One big advantage of using a GUMS-server is, that it can centralise and take over many tasks, like querying the VOMS-servers.

“xacml-vo-mapping”-Plug-in

This plug-in allows the usage of the e**X**tensible **A**ccess **C**ontrol **M**arkup **L**anguage (XACML) within gPlazma[10], which is typically done, when a GUMS- (Grid User Management System) or SCAS-server (**S**ite **C**entral **A**uthorization **S**ervice) should be used for providing mapping-services.

The plug-in is configured via the following parameters in the “gPlazma-policy-configuration-file”:

- `XACMLmappingServiceUrl`
  Specifies the URL of the mapping-service (typically the GUMS- or SCAS-server).
- `xacml-vo-mapping-cache-lifetime`
  Specifies the caching lifetime of queried mappings in seconds. If set to `0` caching is disabled.
- `gridVoRoleStorageAuthzPath`[8]
  Specifies the pathname of the “storage-authzdb-style”-file to be used by the plug-in, which is only required when using the so called “GUMSAuthorization-ServicePort”, but not when using the “StorageAuthorizationServicePort”.

“xacml-vo-mapping”-Plug-in

The basic procedure of using a GUMS- or SCAS-server is the same as described above on slide 19.

For the SCAS-server there is however one important difference:
An SCAS-server returns a user-ID, a group-ID and zero or more secondary-group-IDs.
gPlazma concatenates the user-ID and group-ID separated by a “ :” and the actual mappings are determined by taking this as virtual user-name in the plug-ins’ “storage-authzdb-style”-file.

Mapping with “storage-authzdb-Style”-Files

“storage-authzdb-style”-files are used by several plug-ins for mapping virtual user-names to the actual UNIX user-ID(s) [4] and group-IDs [4] (as they are also used by dCache’s file-hierarchy-provider, which is either Chimera or the legacy PNFS) as well as some other information.

There are two ways of mapping that can be used in a “storage-authzdb-style”-file:

- static mapping-method
  With this method the “storage-authzdb-style”-file directly specifies the data to which a virtual user-name is mapped.
- dynamic mapping-method
  With this method the “storage-authzdb-style”-file specifies the names of built-in functions, that handle the mapping to a user-ID [4] and to group-IDs [4] respectively. The other information is still directly mapped via the “storage-authzdb-style”-file.

Currently this is used by the grid-mapfile-, gplazmalite-vorole-mapping- and in some cases by the saml-vo-mapping- and xacml-vo-mapping-plug-ins.

“storage-authzdb-Style”-File Syntax and Semantics

Each line starts with a keyword and is interpreted according to the currently set “storage-authzdb-style”-version or its default [11].
Keywords are followed by their respective arguments, which are all separated by white-space.
Lines not starting with a keyword are currently ignored.

The following keywords are recognised:

- `version`
  Sets the current “storage-authzdb-style”-version, that will be used when interpreting entries.
  It will be valid until the next occurrence of a `version`-keyword.
  The following versions are currently available:
  - `2.1`
    The default version.
  - `2.2`
    This version adds support for priorities of entries [12].

“storage-authzdb-Style”-File Syntax and Semantics

- `authorize`
  Introduces a mapping with the static mapping-method, using the following syntax and semantics:
  `authorize` *`virtual_user-name access-mode`* {*`priority`*}[version 2.2] *`user-ID group-ID`*[`,`*`group-ID`*]* *`home-path root-path fs-root-path`*

  `access-mode` specifies the allowed access-mode for the respective virtual user-name and can be either set to `read-write` or to `read-only`.
  `priority` specifies the priority of the entry using non-negative integers, where a higher number means a higher priority. It cannot be set with the 2.1 [13]- but must be set with the 2.2“- storage-authzdb-style”-version.

  Multiple *`group-ID`*s are separated by “`,`”.

“storage-authzdb-Style”-File Syntax and Semantics

`home-path` is of legacy use and specified the home-directory for a virtual user-name. It is always interpreted relatively to `root-path`.

`root-path` is of legacy use and specified the root-directory [14] for a virtual user-name. It is always interpreted as an absolute path.

`fs-root-path` is of legacy use and was needed for Kerberos.

These paths should not be used anymore, especially as they are not necessarily respected by all door-cells.
However, a value must be specified and it is suggested to set all three to “`/`”.

“storage-authzdb-Style”-File Syntax and Semantics

- `dynamic`
  Introduces a mapping with the dynamic mapping-method, using the following syntax and semantics:
  `dynamic` *`virtual_user-name`* *`access-mode`* {*`priority`*}[version 2.2]
  `user-ID-function group-ID-function home-path root-path fs-root-path`

  `access-mode` specifies the allowed access-mode for the respective virtual user-name and can be either set to `read-write` or to `read-only`.
  `priority` specifies the priority of the entry using non-negative integers, where a higher number means a higher priority. It cannot be set with the 2.1 [13]- but must be set with the 2.2“- storage-authzdb-style”-version.

  `user-ID-function` and *`group-ID-function`* are specifying the names of the built-in functions that return the actual UNIX user-ID(s) [15] and group-ID(s) [15] (as they are also used by dCache’s file-hierarchy-provider, which is either Chimera or the legacy PNFS) respectively.

“storage-authzdb-Style”-File Syntax and Semantics

home-path is of legacy use and specified the home-directory for a virtual user-name. It is always interpreted relatively to *root-path*.
root-path is of legacy use and specified the root-directory [14] for a virtual user-name. It is always interpreted as an absolute path.
fs-root-path is of legacy use and was needed for Kerberos.

These paths should not be used anymore, especially as they are not necessarily respected by all door-cells.
However, a value must be specified and it is suggested to set all three to “ /”.

“storage-authzdb-Style”-File Syntax and Semantics

Example:

```
version 2.1

authorize atlas001 read-only 1000 100 / / /
authorize prdatl01 read-write 1001 101 / / /

dynamic atlas_map read-write dn_uidmap role_gidmap / / /
```

Functions for the Dynamic Mapping-Method

These functions are used to handle the mapping to user-ID(s) [15] and group-ID(s) [15], where it depends on the specific function whether multiple values are returned or not. The virtual user-name is not necessarily used for the actual mapping.

Currently dCache provides the following functions for use with the dynamic mapping-method:

- `dn_uidmap`
  Maps the DN (that is “currently” evaluated by gPlazma) to an actual UNIX user-ID (as it is also used by dCache’s file-hierarchy-provider, which is either Chimera or the legacy PNFS), using the mappings that were specified in `/etc/grid-security/grid-uidmap`.
- `role_gidmap`
  Maps the FQAN (that is “currently” evaluated by gPlazma) to an actual UNIX group-ID [16] (as it is also used by dCache’s file-hierarchy-provider, which is either Chimera or the legacy PNFS), using the mappings that were specified in `/etc/grid-security/grid-gidmap`.

“grid-uidmap-Style”-File Syntax and Semantics

Each line specifies a mapping from a client-certificate’s DN to an actual UNIX user-ID (as it is also used by dCache’s file-hierarchy-provider, which is either Chimera or the legacy PNFS) via the following syntax:

`"`*`distinguished_name`*`" `*`user-ID`*

`distinguished_name` must always be quoted using “ `"`”.
Lines not starting with “ `"`” are currently ignored by dCache.
If the same DN occurs in multiple lines then only the mapping from the last one is used.

Examples:

- `"/C=DE/O=GermanGrid/OU=LMU/CN=Christoph Anton Mitterer" 1000`
- `"/C=DE/O=GridGermany/OU=Leibniz-Rechenzentrum/CN=Christoph Anton Mitterer" 1001`

“grid-gidmap-Style”-File Syntax and Semantics

Each line specifies a mapping from a client-certificate’s FQAN to an [16] actual UNIX group-ID (as it is also used by dCache’s file-hierarchy-provider, which is either Chimera or the legacy PNFS) via the following syntax:

`"`*`fqan`*`" `*`group-ID`*

`fqan` must always be quoted using “ `"`”.
Lines not starting with “ `"`” are currently ignored by dCache.
If the same FQAN occurs in multiple lines then only the mapping from the last one is used.

Examples:

- `"/atlas" 100`
- `"/atlas/de" 110`
- `"/atlas/Role=production" 101`

Overview on Using the `dn_uidmap`- and `role_gidmap`-Functions

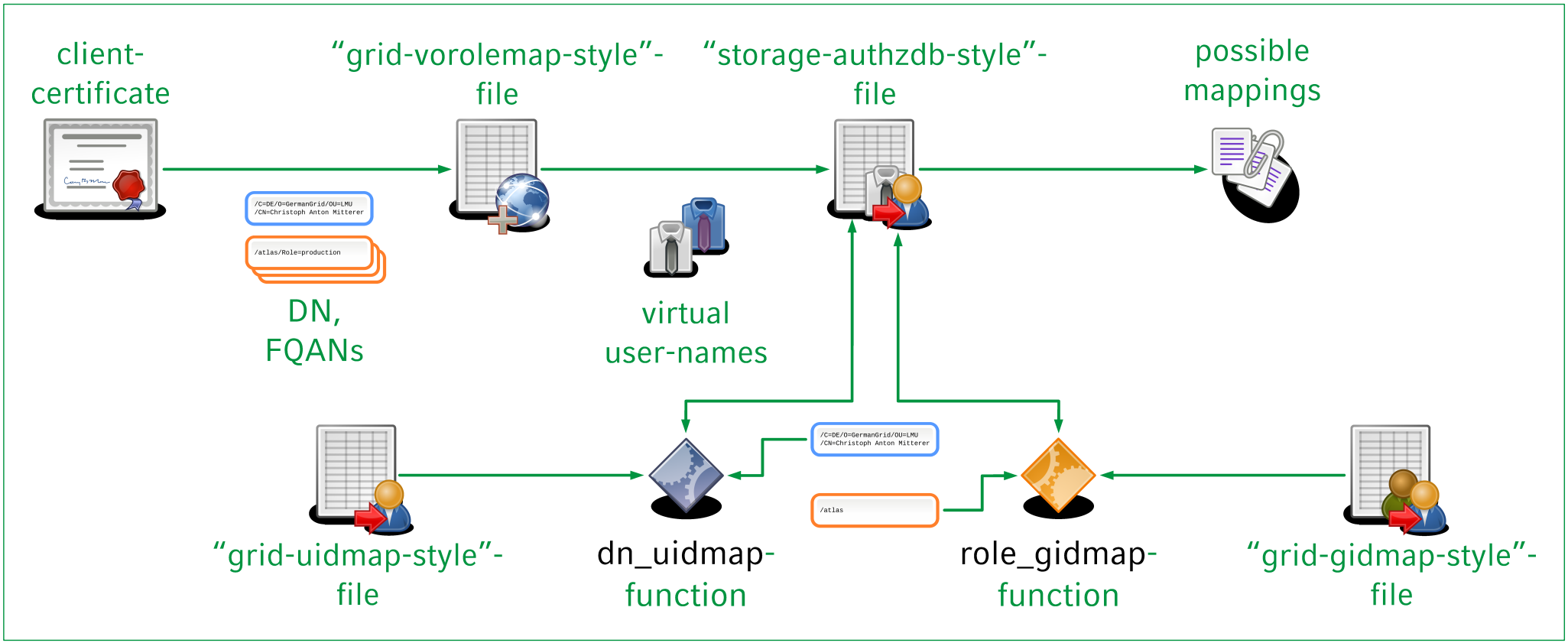
The idea behind the `dn_uidmap`- and `role_gidmap`-functions is, that the DN and FQANs of an entity making an access-request, should be mapped independently to one user-ID and zero or more group-IDs respectively.

In other words:

- Each DN corresponds exactly one user-ID.
- Each FQAN (and thus for example each role) corresponds to one group-ID [16].

Overview on Using the `dn_uidmap`- and `role_gidmap`-Functions

The following illustration describes the whole process using the “gplazmalite-vorole-mapping”-plug-in as example:



Overview on Using the `dn_uidmap`- and `role_gidmap`-Functions

Examples for the configuration-files:

- "grid-vorolemap-style"-file:

```
"*" "/atlas" atlas_map
"*" "/atlas/de" atlas_map
"*" "/atlas/Role=production" atlas_map
```

- "storage-authzdb-style"-file:

```
dynamic atlas_map read-write dn_uidmap role_gidmap / / /
```

- `/etc/grid-security/grid-uidmap`:

```
"/C=DE/O=GermanGrid/OU=LMU/CN=Christoph Anton Mitterer" 1000
"/C=DE/O=GridGermany/OU=Leibniz-Rechenzentrum/
CN=Christoph Anton Mitterer" 1001
```

- `/etc/grid-security/grid-gidmap`:

```
"/atlas" 100
"/atlas/de" 110
"/atlas/Role=production" 101
```

III. Access Control Subprocesses

Not yet finished, will be made available later.

IV. Access Control Lists

Overview on Access Control Lists

Similar to UNIX-like-operating-systems, dCache provided originally only a limited set of permissions, that could be set on each file (regular files and directories).

This set of permissions corresponds to the traditional POSIX file permission modes:

- read, write and execute permissions for the owning user
- read, write and execute permissions for the owning group
- read, write and execute permissions for others

Eventually, a more powerful system supporting permissions for different users and groups was demanded by dCache's users, especially those from the LHC-community.

Such extended permissions are generally called ACLs (**A**ccess **C**ontrol **L**ists), which allow whole lists of so called ACEs (**A**ccess **C**ontrol **E**ntries) to be set on files. Each ACE specifies a permission for its file.

In dCache, ACLs were introduced starting with version 1.9.3.

Overview on Access Control Lists

Some well-known ACL-systems include:

- POSIX.1e draft 17 ACLs
- Network File System version 4 ACLs
- Windows ACLs

Motivation for Access Control Lists

General reason for ACLs is the need to set more powerful and complex permissions on files, than the traditional POSIX file permission modes would allow.

From the perspective of LCG Storage Elements this can be demonstrated on the following scenario:
Within the ATLAS-experiment, data is created and distributed either manually by users or automatically by some central “production-services”. Each of them can “clean-up” his own storage-areas, but normal users should usually not be able to write to “production-storage-areas”.
So far this can be fully achieved with traditional POSIX file permission modes.

Additionally it is desired, that special “deletion-services” can clean-up the storage-areas (for example volatile storage-areas) of normal users, too.
This is however not longer possible with traditional POSIX file permission modes, as two different entities would need write permissions to the same files.

The same problem was for example seen with the “ATLASLOCALGROUPDISK”-storage-area.

Access Control Lists in dCache

As explained before, gPlazma returns a number of actual UNIX user-ID(s) [4] and group-IDs [4] as well as some other information to the invoking door-cell. The door-cell evaluates this data against the ACLs and the traditional POSIX file permission modes in order to determine whether the access should be granted or not.

Door-cells using gPlazma, support ACLs in the following ways:

- GSIdcap-door-cells
  Fully support ACLs in addition to traditional POSIX file permission modes.
- GridFTP-door-cells
  Fully support ACLs in addition to traditional POSIX file permission modes.
- SRM-door-cells
  Directly only support traditional POSIX file permission modes (used for deletion of files and creation of directories) at the moment.

  Indirectly ACLs are “fully supported”, as SRM “delegates” the actual data-transfer to one of the above protocols.

Access Control Lists in dCache

For door-cells not using gPlazma the situation with support for ACLs is the following:

- dcap-door-cells
  Fully support ACLs in addition to traditional POSIX file permission modes.
  However, entities are always [17] mapped to the user `nobody` and to the group `nogroup`, thus making the use of ACLs pointless.
- xrootd-door-cells
  Implement the so called “ALICE security model” which provides an own system for ACLs.
  Unlike with other ACL-systems, the ACLs are here not stored within the file-system, but in a central catalogue.
  xrootd-door-cells fully support the ACL-system from the “ALICE security model”.
  It is however not covered by this chapter.

Access Control Lists in dCache

dCache's "native" ACLs (not the ones from the "ALICE security model") are a subset of the NFS version 4 ACLs, providing nearly all [18] of their features. They are evaluated in addition to the traditional POSIX file permission modes, which they generally outvote.

The ACLs are stored in the database used by dCache's file-hierarchy-provider. They are intended to be used with Chimera, however, apart from some features [19] it is in principle also possible to use them with the legacy PNFS.

Enabling Access Control Lists and General Configuration

In order to enable the ACL-system, the following steps must be done:

1. If the dCache-cluster is not a “fresh” installation from version 1.9.3 or higher, the SQL-schemes at *dCache-home*`/libexec/chimera/sql/create-dCacheACL.sql`[20] and *dCache-home*`/libexec/chimera/sql/pgsql-procedures.sql` must be imported into a database, typically the same one as used by Chimera.
2. The `permissionHandler`-configuration-parameter must be set to `diskCacheV111.services.acl.ACLPermissionHandler`,`diskCacheV111.services.acl.UnixPermissionHandler` or `diskCacheV111.services.acl.ACLPermissionHandler` in the configuration-file of each door-cell that should use ACLs.
3. The Chimera-cell as well as each door-cell that uses ACLs must be able to communicate with the database where the ACL-tables are stored. The required parameters to configure this are described below.
4. The Chimera-cell and the specific door-cells must be restarted.

Enabling Access Control Lists and General Configuration

The following parameters specify how a cell communicates with the DBMS that holds the ACL-tables and are available in the configuration-files for the Chimera-cell as well as each type of door-cell that supports ACLs.

- `aclConnDriver`
  Specifies the JDBC-driver.
- `aclConnUrl`
  Specifies the JDBC-connection-URL
- `aclConnUser`
  Specifies the user-name to be used for accessing the database.
- `aclConnPswd`
  Specifies the password to be used for accessing the database.
- `aclTable`
  Specifies the name[21] of the ACL-table.

Enabling Access Control Lists and General Configuration

The following parameters specify how a door-cell that supports ACLs makes use of them:

- `permissionHandler`
  Specifies the “permission-handler” to be used by the door-cell.
  The following values are possible:
  - `diskCacheV111.services.acl.UnixPermissionHandler`
    Only the traditional POSIX file permission modes are used.
  - `diskCacheV111.services.acl.ACLPermissionHandler`
    Only the access control lists are used.
  - `diskCacheV111.services.acl.ACLPermissionHandler,diskCacheV111.services.acl.UnixPermissionHandler`
    Access control lists are used at first, but when they do not specify whether a given access has to be granted or denied, the traditional POSIX file permission modes are used.

Structure of Access Control Lists and Access Control Entries

In addition to the traditional POSIX file permission modes, every file (regular files and directories) can have exactly one access control list “attached” to it.

An access control list consists of one or more access control entries in a given order, where each entry specifies a permission for its file.
It is important to understand, that the order of entries can have great influence on the effective permissions and also that entries can be in “conflict”.

An access control entry consists of four fields, namely its type, flags, principal and permissions.

More information about access control lists and access control entries can be found on the `nfs4_acl(5)` manual page of the `nfs4-acl-tools`-package.
It should however be noted, that only the ACE-types, -flags, -principals and -permissions described in this chapter, are currently supported by dCache.

Access Control Entry Type

ACE types are used to specify the “global” meaning and effect a given ACE.

Name	Symbol	Description
ACCESS_ALLOWED	A	An “allow-ACE” allows its principal to perform any access that requires one of its permissions.
ACCESS_DENIED	D	An “deny-ACE” denies its principal to perform any access that requires one of its permissions.

Within the acladmin-cell another set of symbols is used to specify the ACL type:

Symbol	Symbol used within the acladmin-cell
A	+
D	-

Access Control Entry Flags

ACE flags are used to set group- and inheritance-flags on a given ACE.
An allow- or deny-ACE may contain zero or more ACE flags.

As ACEs are inherited from a parent directory's ACL, the inheritance-flags can only be used in ACEs of directories.

Access Control Entry Flags

Name	Symbol	Description
Inheritance-Flags		
FILE_INHERIT	f	Newly created regular files will inherit the ACE, but with its inheritance-flags removed. If the DIRECTORY_INHERIT-flag is not set, newly created subdirectories will also inherit the ACE, but with the INHERIT_ONLY-flag added.
DIRECTORY_INHERIT	d	Newly created subdirectories will inherit the ACE.
INHERIT_ONLY	o	The ACE will not be considered for permission checks, but it will be inherited with its INHERIT_ONLY-flag removed.

Access Control Entry Flags

Name	Symbol	Description
Groups-Flags		
`IDENTIFIER_GROUP`	g	Indicates that the principal represents a group instead of an user.

Access Control Entry Principal

ACE principals are used to specify the entities to which a given ACE applies.

Name	Description
user-name	Any entity which is mapped (for example by gPlazma) to the same actual UNIX user-ID as *user-name* is.
group-name	Any entity which is mapped (for example by gPlazma) to the same actual UNIX user-ID as *group-name* is.
OWNER@	Any entity which is mapped (for example by gPlazma) to the same actual UNIX user-ID as the owning user of the regular file or directory is.
GROUP@	Any entity which is mapped (for example by gPlazma) to the same actual UNIX group-ID as the owning group of the regular file or directory is.
EVERYONE@	Any entity.

Access Control Entry Principal

Name	Description
ANONYMOUS@	Any non-authenticated entity (the opposite of AUTHENTICATED@).
AUTHENTICATED@	Any authenticated entity (the opposite of ANONYMOUS@).

Within the acladmin-cell another set of symbols is used to specify the ACL principal:

Name	Name used within the acladmin-cell
user-name	USER:*user-ID*
group-name	GROUP:*group-ID*

Access Control Entry Principal

It should be noted, that the `OWNER@`-, `GROUP@`-, and `EVERYONE@`-principals are not fully identical with the “user”, “group” and “other” classes from the traditional POSIX file permission modes.

“group” is only used when “user” does not match and “other” is only used when neither “user” nor “group” matches.
The principals however work on a “use-the-first-one-that-matches”-basis (thus `EVERYONE@` “includes” `OWNER@` and `GROUP@`.

Access Control Entry Permissions

ACE permissions are used to set different types of permissions on a given ACE.
An ACE should contain one ore more ACE permissions.

The permissions may have a different semantic meaning depending on the file-type (regular file or directory).
Some permissions may only be usable with a specific file-type.

Access Control Entry Permissions

Name	Symbol	Description
READ_DATA	r	Permission to read from a regular file.
(LIST_DIRECTORY)	(l)	Permission to list a directory.
WRITE_DATA	w	Permission to write to a regular file.
(ADD_FILE)	(f)	Permission to create regular files in a directory.
APPEND_DATA	a	Permission to append data to a regular file.
(ADD_SUBDIRECTORY)	(s)	Permission to create subdirectories in a directory.
EXECUTE	x	Permission to execute a regular file.
		Permission to change to a directory.
DELETE	d	Permission to delete the regular file or directory. [21]
DELETE_CHILD	D	Permission to delete a regular file or subdirectory from a directory. [22]

Access Control Entry Permissions

Name	Symbol	Description
READ_ATTRIBUTES	t	Permission to read the attributes of the regular file or directory.
WRITE_ATTRIBUTES	T	Permission to write the attributes of the regular file or directory.
READ_NAMED_ATTRS	n	Permission to read the named attributes of the regular file or directory.
WRITE_NAMED_ATTRS	N	Permission to write the named attributes of the regular file or directory.
READ_ACL	c	Permission to read the ACL of the regular file or directory.
WRITE_ACL	C	Permission to write the ACL of the regular file or directory.

Access Control Entry Permissions

Name	Symbol	Description
WRITE_OWNER	o	Permission to write the owning user and group of the regular file or directory.

Inheritance of Access Control Entries

In contrast to POSIX.1e draft 17 ACLs, with NFS version 4 ACLs it is possible to specify rules for inheritance of access control entries.

Inheritance is controlled by using the `FILE_INHERIT`-, `DIRECTORY_INHERIT`-, and `INHERIT_ONLY`-flags on ACEs of directories, as described above.
In addition, the following notes shall be given:

- ACEs are generally only inherited from directories and not from regular files.
- A newly created regular file or directory inherits the ACEs marked for inheritance from its parent directory's ACL.
- ACEs are only inherited once at the creation-time of regular file or directory and not again later.
- Even if `DIRECTORY_INHERIT` is not set in a parent directory's ACE, it can be inherited by a newly created subdirectory, when `FILE_INHERIT` was set on it. In this case, `INHERIT_ONLY` will be added to the inherited ACE.

Inheritance of Access Control Entries

- dCache generally uses so called “propagating inheritance”, which means that the `FILE_INHERIT`- and `DIRECTORY_INHERIT`-flags are not removed on newly created subdirectories.
  The `INHERIT_ONLY`-flag is however removed on newly created subdirectories.
- ACEs having the `INHERIT_ONLY`-flag set are never considered for permissions checks (on its directory).
- Having the `INHERIT_ONLY`-flag set on an ACE while neither the `FILE_INHERIT`- nor the `DIRECTORY_INHERIT`-flag is set, is an error.

Evaluation of Access Control Lists

In order to determine whether an access-request should be granted or denied, a door-cell evaluates the access control lists and the traditional POSIX file permission modes, against the possible mappings of an entity, as returned for example by gPlazma.

Depending on the chosen permission-handler, the following happens:

- `diskCacheV111.services.acl.UnixPermissionHandler`
  Only the traditional POSIX file permission modes are evaluated.
- `diskCacheV111.services.acl.ACLPermissionHandler`
  Only the access control lists are evaluated.
  If an ACL results in “undefined” its actual result is “deny”.

Evaluation of Access Control Lists

- `diskCacheV111.services.acl.ACLPermissionHandler,diskCacheV111.services.acl.UnixPermissionHandler`
  For every possible mapping (as returned for example by gPlazma), the access control lists are evaluated:
  - The first time “allow” is returned, this will be used as global result.
  - If always “deny” is returned, this will be used as global result.
  - If always either “deny” or “undefined” is returned, the traditional POSIX file permission modes are evaluated, which always give an explicit result (“allow” or “deny”).

Evaluation of Access Control Lists

The evaluation of an ACL works like the following:

1. If no ACL is present at all for the specific file, the evaluation stops and the result is “undefined”.
2. An access-request requires one or more permissions. In order to check them, the ACL is evaluated ACE-by-ACE in their given order.
   The principal is taken from the currently evaluated mapping (out of the “possible mappings”).
3. All required permissions must be explicitly allowed for the result being “allow”.
   If a single required permission is explicitly denied, the evaluation stops immediately and the result is “deny”.
   If a single required permission is neither explicitly allowed nor denied, the result is “undefined”.
4. Each permission is only checked once. This means, that after a permission has been explicitly allowed or denied it is not further checked within the evaluation of the same ACL.
   Thus, the first allowed- or denied-result “counts” for that single permission.

Evaluation of Access Control Lists

The evaluation of the traditional POSIX file permission modes works as described in the POSIX-standard and as it is well known from any UNIX-like-operating-system.

Evaluation of Access Control Lists

It is important to understand the following:

- The order of ACEs can have a great influence on the effective permissions, especially but not only when using deny-ACEs in the same ACL.
- During evaluation of a given ACL, the first ACE that explicitly allows or denies a permission is used for that single permission. There is neither a “deny-overrides”- nor an “allow-overrides-mechanism”.
- NFS version 4 ACLs are using a “default-deny-mechanism”, which means that if a permission is not explicitly allowed it is denied.
  In dCache, but only if the `diskCacheV111.services.acl.UnixPermissionHandler`-permission-handler is used as fall-back, the result of those permissions is “undefined” allowing the fall-back-permission-handler to determine the global result.

If no fall-back-permission-handler is used, it is suggested to try to avoid deny-ACEs.

Editing Access Control Lists

There are two ways to edit the ACL of a given regular file or directory:

- using `nfs4_getfacl` and `nfs4_setfacl` from the `nfs4-acl-tools`-package
  Detailed documentation can be found in the `nfs4_getfacl(1)` and `nfs4_setfacl(1)` manual pages of this package.
  It should be noted, that this requires a recent enough kernel and tool-set.
- using the dCache administration interface to the acladmin-cell
  Detailed information can be found by using the `help`-command within the administration interface.

V. Examples and Exercises

Introduction

The following assumptions are made within the following examples and exercises:

- A working and "fully" configured dCache installation of version 1.9.3 or higher is used.
- Chimera is used as file-hierarchy-provider.
- PostgreSQL is used as DBMS.
- The student has a basic understanding about dCache and some fundamental Grid-tools (for example the VOMS-Proxy-Certificate-Tools).
- If the dCache installation is a cluster spread over multiple hosts, the student knows when which host and if multiple hosts has to be used for a given step.
- The student has read and understand the previous chapters of this presentation.

Furthermore, 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 "`>`" are entered within dCache's administration interface.
- Standard input is written black, standard output grey and standard error red.

1. Access Control Lists

E1: Enabling Access Control Lists

The goal of this example is to import the required SQL-schemes in the database and to enable the usage of ACLs for all supporting door-cells.

1. Stop the dCache-cluster and Chimera:
   ```
   # dCache-home/bin/dcache stop
   # dCache-home/bin/dcacheChimeraNfs stop
   ```
2. If the dCache-cluster is not a “fresh” installation from version 1.9.3 or higher, import the the required SQL-schemes using the `psql`-command:
   ```
   # sudo -u postgres  psql -f dCache-home/libexec/chimera/sql/create-dCacheACL.sql chimera
   # sudo -u postgres  psql -f dCache-home/libexec/chimera/sql/pgsql-procedures.sql chimera
   ```
   [20]
3. In *dCache-home*`/config/dCacheSetup`, set the `permissionHandler`-parameter:
   ```
   permissionHandler=diskCacheV111.services.acl.ACLPermissionHandler,diskCacheV111.services.acl.UnixPermissionHandler
   ```
4. Set the parameters for the communication with the DBMS as necessary.
5. Start the dCache-cluster and Chimera:
   ```
   # dCache-home/libexec/chimera/chimera-nfs-run.sh start
   # dCache-home/bin/dcache start
   ```

E1: Enabling Access Control Lists

6. Check whether dCache is running correctly and whether its file-hierarchy is mounted correctly.

If everything has worked, all door-cells should use ACLs now and fall back to traditional POSIX file permission modes.

E2: Exploring the Administration Interface to the acladmin-cell

The goal of this exercise is to explore the administration interface to the acladmin-cell and to become familiar with it.

1. Connect to the administration interface to the acladmin-cell:
```
$ ssh -c blowfish -p 22223 admin@localhost
> cd acladmin
```
2. The `help`-command can be used to get an overview on all available commands or detailed information for specific commands:
```
> help
> help getfacl
> help setfacl
```
3. Read the documentation to the `getfacl`- and `setfacl`-commands.
4. Optionally, try out the `getfacl`- and `setfacl`-commands on existing files:
```
> setfacl /pnfs/dcache.org/data/test-file2 OWNER@:+rw
> getfacl /pnfs/dcache.org/data/test-file2
> setfacl 00006D7A29890A194624B78155603E977BF9 AUTHENTICATED@:+rwdtTnNcCo
> getfacl 00006D7A29890A194624B78155603E977BF9
```

It is expected that you well familiar with the syntax of `getfacl`- and `setfacl`-commands. If not repeat step 3.

E3: Simple ACLs – Allowing “Read-Access”

The goal of this exercise is to show a simple ACL, which allows “read-access” for a file’s owning user.

1. Set up the traditional POSIX file permission modes of a regular file to deny any access:
```
# chmod a= /pnfs/dcache.org/data/test-file3
```
2. Trying to read the regular file should give an authorisation-error:
```
$ globus-url-copy gsiftp://localhost/pnfs/dcache.org/data/test-file3 file://"$
{PWD}"/downloaded-test-file3
error: globus_ftp_client: the server responded with an error
550 Permission denied
```
3. Add an ACL for the regular file that permits its owning user to read:
```
> setfacl /pnfs/dcache.org/data/test-file3 OWNER@:+r
```
4. Repeat step 2, which should succeed now.

Of course, this alone would also be possible with traditional POSIX file permission modes.

E4: Simple ACLs – Denying “Read-Access”

The goal of this exercise is to show a simple ACL, which denies “read-access” for a specific user.

1. Set up the traditional POSIX file permission modes of a regular file to allow any read-access:
   ```
   # chmod a=r /pnfs/dcache.org/data/test-file4
   ```
2. Trying to read the regular file should work:
   ```
   $ globus-url-copy gsiftp://localhost/pnfs/dcache.org/data/test-file4 file://"${PWD}"/downloaded-test-file4
   ```
3. Add an ACL for the regular file that denies a specific user to read:
   ```
   > setfacl /pnfs/dcache.org/data/test-file4 USER:user-ID:-r
   ```
4. Repeat step 2, which should give an authorisation-error now:
   ```
   error: globus_ftp_client: the server responded with an error
   550 Permission denied
   ```

Using traditional POSIX file permission modes, it would only be possible to deny read-access for a single user (namely the owning user) or a single group (namely the owning group).

E5: Simple ACLs – Further Exercises

The goal of this exercise is to further explore and try out some simple ACLs.

1. Try to set up an ACL for a regular file, that allows it to be read by any authenticated entity, to be written and deleted by the owning user, to be appended by the owning group and to be deleted by some individual user.
   ```
   > setfacl /pnfs/dcache.org/data/test-file5 AUTHENTICATED@:+r OWNER@:+wd GROUP@:user-ID:+a USER:user-ID:+d
   ```
2. Try to set up an ACL for a directory, that allows it only to be listed by its owning user and group.
   ```
   > setfacl /pnfs/dcache.org/data/test-directory5a USER@:+l GROUP@:+l
   ```
3. Try to set up an ACL for a directory, that allows it to be listed and changed to by everyone, where only the owning user can delete regular files and directories and where only the owning user and group can create new regular files and subdirectories.
   ```
   > setfacl /pnfs/dcache.org/data/test-directory5b OWNER@:+fsD GROUP@:+fs EVERYONE@:+lx
   ```
4. Think about whether the order of the above ACLs matters or not.
5. Think about whether this is influenced by the file's traditional POSIX file permission modes. If so how could this be avoided?

E6: Order of ACEs

The goal of this exercise is to show that the order of ACEs can have a great influence on the effective permissions.

1. Set up the traditional POSIX file permission modes of a regular file to deny any access:
```
# chmod a= /pnfs/dcache.org/data/test-file6
```
2. Add the following ACL to a regular file:
```
> setfacl /pnfs/dcache.org/data/test-file6 OWNER@:+r OWNER@:-r
```
3. Trying to read the regular file should work:
```
$ globus-url-copy gsiftp://localhost/pnfs/dcache.org/data/test-file6 file://"${PWD}"/downloaded-test-file6
```
4. Switch the order of ACEs:
```
> setfacl /pnfs/dcache.org/data/test-file6 OWNER@:-r OWNER@:+r
```
5. Repeat step 2, which should give an authorisation-error now:
```
error: globus_ftp_client: the server responded with an error
550 Permission denied
```

This demonstrates that even a simple swapping of rules totally changes the effective permissions.

E7: ACLs and Multiple Possible Mappings

The goal of this exercise is to demonstrate some simple ACLs when the client's certificate leads to multiple possible mappings.

In this example, the client's certificate is supposed to be able to have multiple FQANs (namely one with the "production"- and one with the "admin"-role). Thus the possible mappings (as returned for example by gPlazma) can lead to exactly one user-ID and up to 3 group-IDs.

Different FQANs on the VOMS-proxy-certificate can be created with:

```
# voms-proxy-init --voms dcache
# voms-proxy-init --voms dcache:dcache/Role=production
# voms-proxy-init --voms dcache:dcache/Role=production --voms dcache:dcache/Role=admin
```

1. Set up the traditional POSIX file permission modes of three regular files to deny any access:

   ```
   # chmod a= /pnfs/dcache.org/data/test-file7a /pnfs/dcache.org/data/test-file7b
   /pnfs/dcache.org/data/test-file7c
   ```

E7: ACLs and Multiple Possible Mappings

2. Set the owning group of two of the files to different groups:

```
# chown :production /pnfs/dcache.org/data/test-file7b
# chown :admin /pnfs/dcache.org/data/test-file7c
```

3. Add an ACL for the regular file that permits its owning group to read, but denies everyone else (including the owning user):

```
> setfacl /pnfs/dcache.org/data/test-file7a GROUP@:+r
> setfacl /pnfs/dcache.org/data/test-file7b GROUP@:+r
> setfacl /pnfs/dcache.org/data/test-file7c GROUP@:+r
```

4. Trying to read the regular files should only work, when the corresponding FQANs are set in your VOMS-proxy-certificate.
5. Optionally, expand your ACLs with deny-ACEs for one of the possible mappings and try reading with different FQANs set in your VOMS-proxy-certificate. Think about why it works or not.

E8: Multiple Matching Principals per evaluated ACL and Deny-ACEs

The goal of this exercise is to demonstrate the effects of deny-ACEs when multiple principals match per evaluated ACE. The client's certificate should have no additional FQANs.

1. Set up the traditional POSIX file permission modes of a regular file to deny any access:
```
# chmod a= /pnfs/dcache.org/data/test-file8
```
2. Add an ACL for the regular file that permits its owning user to read and denies everyone else:
```
> setfacl /pnfs/dcache.org/data/test-file8 EVERYONE@:-r OWNER@:+r
```
3. Trying to read the regular file should give an authorisation-error:
```
$ globus-url-copy gsiftp://localhost/pnfs/dcache.org/data/test-file8 file://"$
{PWD}"/downloaded-test-file
error: globus_ftp_client: the server responded with an error
550 Permission denied
```
4. Switch the order of ACEs:
```
> setfacl /pnfs/dcache.org/data/test-file8 OWNER@:+r EVERYONE@:-r
```
5. Repeat step 3, which should succeed now.
6. Think about, why this happens.

E9: Inheriting ACEs

The goal of this exercise is to demonstrate inheritance of ACEs.

1. Are the following ACLs possible and if not why:
```
> setfacl /pnfs/dcache.org/data/test-file9a OWNER@:+r:fd
> setfacl /pnfs/dcache.org/data/test-file9b GROUP@:+d:i
> setfacl /pnfs/dcache.org/data/test-directory9a OWNER@:+lfsxdD:fdi
> setfacl /pnfs/dcache.org/data/test-directory9b EVERYONE@:-lfs:fd
```
2. Add some ACLs to directories using the different inheritance flags:
```
> setfacl /pnfs/dcache.org/data/test-test-directory9c OWNER@:+r:f
> setfacl /pnfs/dcache.org/data/test-test-directory9d OWNER@:+r:d
> setfacl /pnfs/dcache.org/data/test-test-directory9e OWNER@:+r:fd
> setfacl /pnfs/dcache.org/data/test-test-directory9f OWNER@:+r:fi
> setfacl /pnfs/dcache.org/data/test-test-directory9g OWNER@:+r:di
> setfacl /pnfs/dcache.org/data/test-test-directory9h OWNER@:+r:fdi
```
3. Try to create regular files (using `globus-url-copy`) and subdirectories (using `edg-gridftp-mkdir`) below them.
4. In the administration interface to the acladmin-cell, see how the ACEs were inherited by the regular files and directories.

E10: A Real World Example

The goal of this exercise is to demonstrate the benefits of ACLs using a “real world scenario”. It is similar to the one from the ATLAS-experiment, as described above.

Within a given directory and its subdirectories:

- Everyone should be allowed to read all regular files and to list and change to all directories.
- “Normal” users should be allowed to create new regular files and directories.
- “Normal” users should further be allowed to write to any regular file.
- “Normal” users and those with the “production”-role should be allowed to delete regular files and directories.

It is impossible to realise this with traditional POSIX file permission modes.

1. Create the base-directory:

```
$ edg-gridftp-mkdir gsiftp://localhost/pnfs/dcache.org/data/test-directory10
```

2. Add an ACL for the directory that implements the constraints from above:

```
> setfacl /pnfs/dcache.org/data/test-directory10 EVERYONE@:+r:f EVERYONE@:+lx:d
GROUP:normal-group-ID:+fs:d GROUP:normal-group-ID:+w:f GROUP:normal-group-ID:+d:fd
GROUP:normal-group-ID:+D:d GROUP:production-group-ID:+d:fd GROUP:production-group-ID:
+D:d
```

E10: A Real World Example

3. Try to create new regular files (using `globus-url-copy`) and new subdirectories (using `edg-gridftp-mkdir`) at some levels below the base-directory.
4. In the administration interface to the acladmin-cell, see how the ACEs were inherited by the regular files and directories.
5. Try to list the directories at several levels (using `edg-gridftp-ls`).
6. Try to remove regular files (using `edg-gridftp-rm`) and directories (using `edg-gridftp-rmdir`) as “normal” user and with the “production”-role.
7. Think about why it is impossible to realise this exercise with only traditional POSIX file permission modes. There are at least two important reasons.
8. Think about why inheritance is required.
9. Think about whether it would make sense to use the `INHERIT_ONLY`-flag.

Footnotes

1. Via disabling gPlazma and the `kpwdFile` configuration-parameter in *`dCache-home`*`/config/gridftpdoorSetup` and *`dCache-home`*`/config/srmSetup`.
2. When invoking gPlazma as module, it is even possible to set a separate "gPlazma-policy-configuration-file" for each door-cell via the `gplazmaPolicy` configuration-option in its respective configuration-file (for example *`dCache-home`*`/config/gsidcapdoorSetup`, *`dCache-home`*`/config/gridftpdoorSetup` or *`dCache-home`*`/config/srmSetup`).
3. In the configuration files *`dCache-home`*`/config/gridftpdoorSetup`, *`dCache-home`*`/config/srmSetup` and *`dCache-home`*`/config/utilitySetup`.
4. Whether multiple user-IDs and group-IDs can occur or not, depends in principle on the used plug-in and with some plug-ins on the usage of the "storage-authzdb-style"-file.
   This however, reflects the current possible way.
5. "grid-mapfile-style"-files as defined by the Globus Toolkit documentation would allow more mappings per line. This is however not supported by dCache.

Footnotes

6. VO-attributes and their use in so called “Fully Qualified Attribute Names” is standardised by the EU’s DataGrid project in the “VOMS Credential Format”-document.
7. Actually, the “ `*`” regular expression needs not to be quoted.
8. Currently, this uses the same configuration parameter as the “gplazmalite-vorole-mapping”-plug-in.
9. These must be present in `/etc/grid-security/` and `/etc/grid-security/certificates/`.
10. Using an XACML-profile for Grid Services.
11. The current default “storage-authzdb-style”-version is 2.1.
12. Although gPlazma supports priorities of entries, the concept is currently not used by any type of door-cell.
13. The priority of a 2.1“ - storage-authzdb-style”-version-entry is implicitly set to `0`.
14. Access “above” the user’s root-directory is generally impossible.
15. It depends on the specific built-in function whether multiple values are returned or not.

Footnotes

16. Currently, the `role_gidmap`-function, maps each FQAN to exactly one group-ID. This might however change in the future.
17. This actually depends on whether the access is done via a dcap-URI or locally via a mounted dCache-file-system.
    In the first case, the user is always `nobody` and the group is always `nogroup`.
    This also applies to the second case, but only when “strong-authentication” is activated, which is the default. If it is deactivated, the client can specify any user and group, which is apparently a security-risk.

    TODO
18. dCache’s ACLs are missing some features like “no-propagating inheritance”. An exact list can be found by comparing the ACE-types, -flags, -principals and -permissions described in chapter IV with the specifications from the NFS version 4 ACLs.
19. ACE-inheritance and the inheritance-flags cannot be used with PNFS as dCache’s file-hierarchy-provider.

Footnotes

20 The ACL-tables created by *dCache-home*`/libexec/chimera/sql/create-dCacheACL.sql` will allow old file-IDs from the legacy PNFS to be used, while with the very similar SQL-scheme in *dCache-home*`/libexec/chimera/sql/addACLtoChimeraDB.sql` new file-IDs from Chimera are required.

21 The suggested and default name is `t_acl`.

22 Some NFS version 4 servers will allow to delete the regular file or directory if either `DELETE` is set in its specific ACE or `DELETE_CHILD` is set in the specific ACE of its parent directory.
Currently, dCache allows it with both.

Acknowledgements

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

- von Berg, Anna
- Fuhrmann, Patrick
- Hesselroth, Ted
- Jung, Christopher
- Kozlova, Irina
- Mejia, Jose
- Mkrtchyan, Tigran
- Mol, Xavier
- Synge, Owen
- Tsigenov, Oleg

Finis coronat opus.