



edg-voms-admin



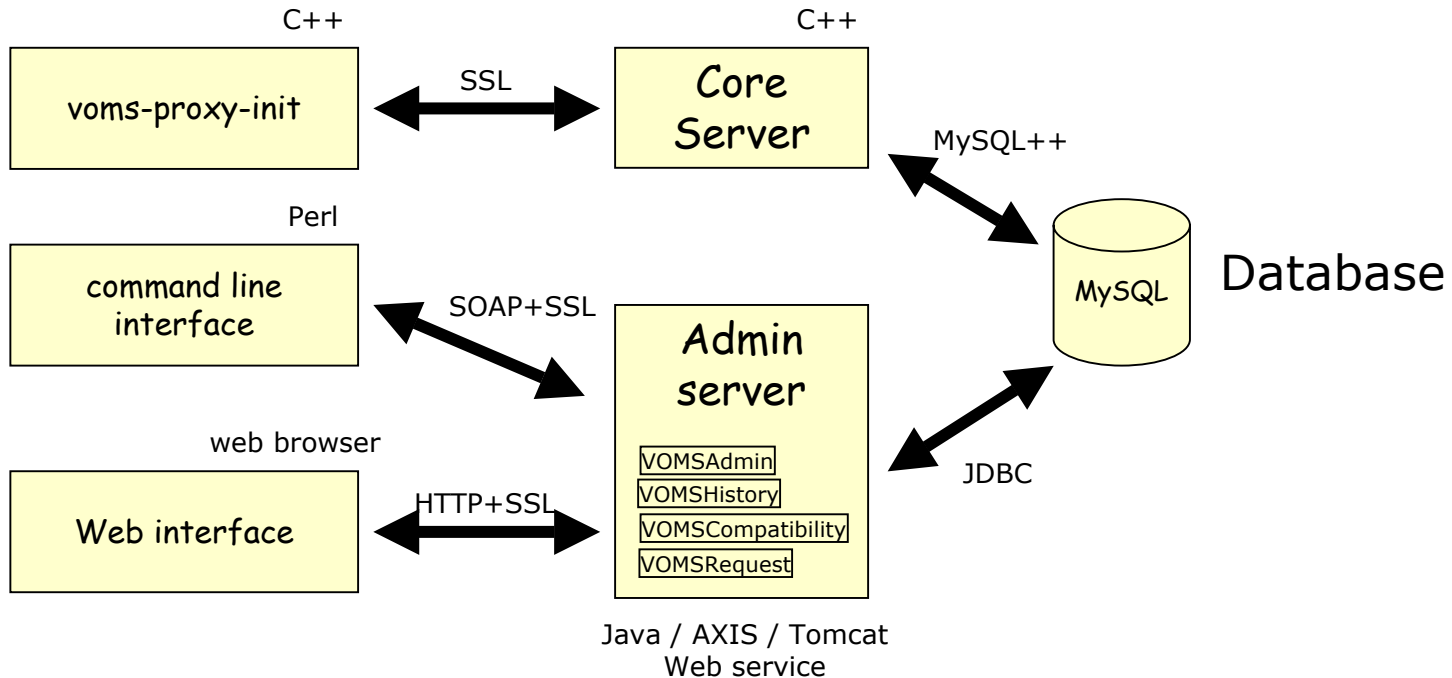
European DataGrid Project

Security Coordination Group

<http://cern.ch/hep-project-grid-scg>

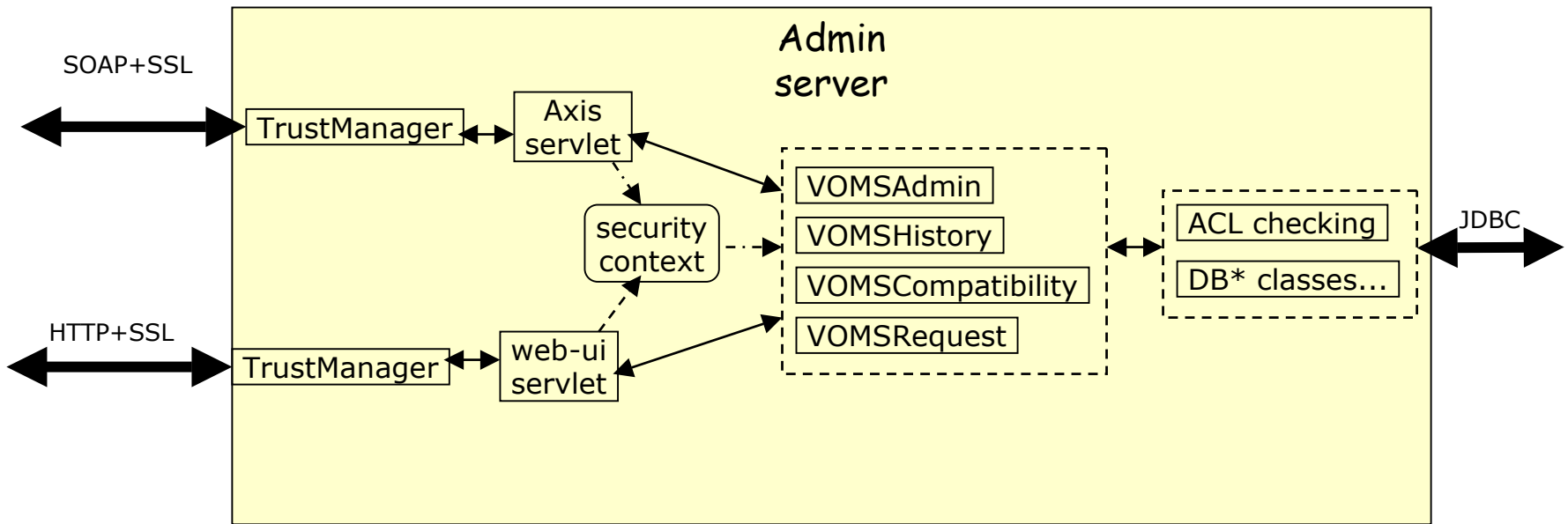
VOMS architecture

- MySQL database - with history and audit records
- core query service and client in C++
- Java Web Service based administration interface
 - Perl client (batch processing)
 - Web browser client (generic administrative tasks)
- replication is planned, but not implemented yet



VOMS internals

- SOAP and HTTP access are sharing the implementation of the business logic
- authentication: Trust Manager (edg-java-security)
- authorization: ACL checking logic inside the implementation





Proxy in Perl client

```
package EDG::HTTPS;
sub setEnv {
    # euid is not 0: this is a user with user cert or proxy
    if($>) {
        $proxy = $ENV{X509_USER_PROXY} || "/tmp/x509up_u" . $>;
        splitProxy();
        $ENV{HTTPS_CERT_FILE} = $proxy. '.proxy';
        $ENV{HTTPS_KEY_FILE} = $proxy. '.key';
        $ENV{HTTPS_CA_FILE} = $proxy. '.user';
        if($@) {
            # fallback solution: the real user certificate
            $ENV{HTTPS_CERT_FILE} = $ENV{X509_USER_CERT} ||
            $ENV{HOME}.'/.globus/usercert.pem';
            $ENV{HTTPS_KEY_FILE} = $ENV{X509_USER_KEY} || $ENV{HOME}.'/.globus/userkey.pem';
        }
    }
    # euid is 0: this is a daemon, so we use the hostcert/key
    else {
        $ENV{HTTPS_CERT_FILE} = $ENV{X509_USER_CERT} || '/etc/grid-security/hostcert.pem';
        $ENV{HTTPS_KEY_FILE} = $ENV{X509_USER_KEY} || '/etc/grid-security/hostkey.pem';
    }
    $ENV{HTTPS_VERSION} = 3;
    $ENV{HTTPS_CA_DIR} = $ENV{X509_CERTDIR} || '/etc/grid-security/certificates';
}
```



SecurityContext

```
package org.edg.security.voms.service;

public class InitSecurityContext extends BasicHandler {

    public static void initSC (ServletRequest req) {
        SecurityContext sc = new SecurityContext();
        SecurityContext.setCurrentContext(sc);

        sc.setClientCertChain((X509Certificate[]) req.getAttribute("javax.servlet.request.X509Certificate"));

        // We use the old DN format in the database.
        if (sc.getClientName () != null)
            sc.setClientName (new DNConvert (sc.getClientName ()).removeProxySuffix (DNConvert.X500));
        if (sc.getIssuerName () != null)
            sc.setIssuerName (new DNConvert (sc.getIssuerName ()).reformat (DNConvert.X500));
    }
}
```



VOMSAdmin.getUser()

```
package org.edg.security.voms.service.admin;

public final class VOMSAdminSoapBindingImpl implements VOMSAdmin {

    synchronized public User getUser (String username, String userca) throws RemoteException {

        try {

            QueryWrapper c = QueryWrapper.get ();

            try {

                DBGroup vo = DBGroup.getVOGroup (c);

                vo.checkPermission (c, Operation.LIST);

                User result = DBUser.getInstance (c, username, DBCA.getInstance (c, userca)).getAsUser (c);

                log.info ("Returned detailed information about user \'" + result.getDN () + "\'");

                return result;

            }

            finally {

                c.release ();

            }

        }

    }

}
```



DBGroup.checkPermission()

```
package org.edg.security.voms.database;

final public class DBGroup implements DBContainer {
    public void checkPermission (Connection c, Operation o) {
        getACL (c).checkPermission (c, o);
    }
    public DBACL getACL (Connection c) {
        return DBACL.getInstance (c, acldId, this);
    }
}
```

```
CREATE TABLE groups ( -- Holds all groups in a VO.
    gid bigint unsigned NOT NULL, -- Internal entity identifier.
    dn varchar(255) NOT NULL, -- Fully Qualified Group Name
    parent bigint unsigned NOT NULL, -- Parent group.
    -- Applied ACL (entries are in 'or' relation).
    aclid bigint unsigned NOT NULL,
    -- Default ACL for a group/role created under this group.
    defaultAclid bigint unsigned NOT NULL,
)
```

```
CREATE TABLE acl ( -- Access Control List for containers
    aid bigint unsigned NOT NULL, -- ACL identifier (common id for all entries in one list)
    adminid bigint NOT NULL,-- Administrator's identifier
    operation smallint NOT NULL,-- Operation on the container
    allow tinyint NOT NULL, -- 0 means deny, 1 means allow
)
```

```
CREATE TABLE admins ( -- List of the administrator users
    adminid bigint NOT NULL, -- Administrator's identifier
    dn varchar(250) NOT NULL, -- the DN of the administrator
    ca smallint unsigned NOT NULL, -- Issuer certificate authority
)
```

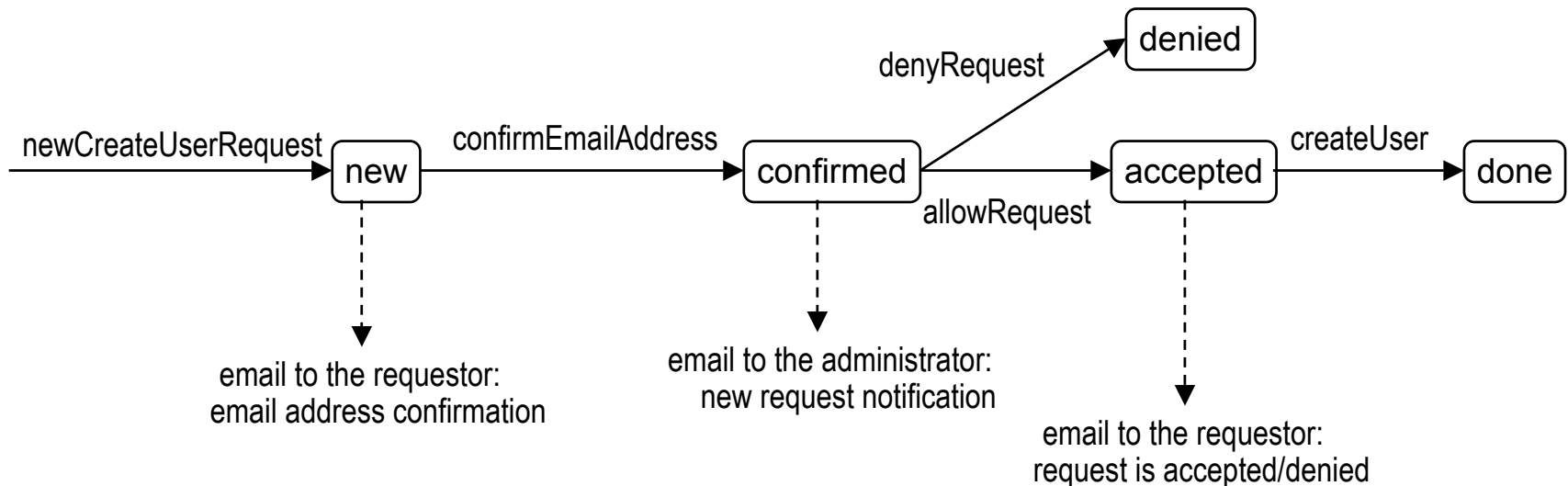

Access Control Lists

container.checkPermission (operation)

- An ACL is a set of *subject-operation-allow* triplets bound to a container
 - The *subject* is (an attribute of) the client requesting the operation
 - If the *allow* flag is true the access is granted (positive match), if false access is denied (negative match)
- *Subject* is matched against all known attributes of the client
 - Client's DN+CA (direct match)
 - Authorization Manager attributes
 - VOMS attributes in the local database
 - VOMS attributes embedded in the client's proxy cert
- Access is granted iff there is a positive match *and* there is no negative match

Request handling (plans)

- VOMSRequest interface for request processing *outside* the core VOMS database tables:
 - new user in the VO
 - member in an existing group
 - ... and for the rest upon request 😊
- planned: extensible Producer/Consumer pattern as an implementation to satisfy complex use cases (e.g. US-CMS VOX project, multiple admins)





Installation

- Install RPMs: `edg-voms-admin`, `edg-voms-admin-interface`, `edg-voms-admin-config` (, `edg-voms-admin-client`)
- `edg-voms-admin-configure install --vo=fred --dbapwd=...`
- `$EDG_LOCATION/etc/init.d/edg-voms-admin start`
- `edg-voms-admin-local fred --add-admin`
- `edg-voms-admin-local fred --add-host`
- CLI:
`edg-voms-admin --url=https://localhost:8443/edg-voms-admin-fred list-users`
- web: `https://localhost:8443/edg-voms-admin-fred/Admin`

VOMS migration

