Pros and cons for using LDAP as backend for an RBAC system

3rd LDAPCon,

Heidelberg, 10.-11.10.2011

Peter Gietz, Markus Widmer,
DAASI International GmbH

Peter.gietz@daasi.de

Markus.widmer@daasi.de

Agenda

- Motivation for Role Based Access Control
- The RBAC standard
- XACML
- OpenRBAC
- Why with OpenLDAP?
- Pros and Cons

Motivation for Role Based Access Control

- Health Level Seven International (HL7) is „the global authority on standards for interoperability of health information technology with members in over 55 countries.“
- Their motivation was:
  - “Simplify authorization management
  - Reduce administrative costs
  - Improve security
  - Enhance partner interoperability
  - Enable new network-level RBAC services”

Motivation for Role Based Access Control

- Another motivation could be compliance (says NIST at http://csrc.nist.gov/groups/SNS/rbac/sarbanes_oxley.html):
  - “The Sarbanes-Oxley Act establishes a set of requirements for financial systems, to deter fraud and increase corporate accountability.
  - For information technology systems, regulators may need to know who used a system, when they logged in and out, what accesses or modifications were made to what files, and what authorizations were in effect.
  - IT vendors responding to Sarbanes-Oxley requirements have adopted RBAC as central to compliance solutions because RBAC was designed to solve this type of problem.”

Motivation for Role Based Access Control

- Using roles makes access control easier and clearly arranged
- When a user changes her role in an organization she automatically has the right privileges for that role
- There are no „special ad hoc solutions“ like:
  - „user X needs permission to access resource Y now, please make it so“
  - Things like this tend not to be documented and thus will be forgotten, even after user X has left the organization
- The real-world organizational structure can be mapped in the role model
  - Changes in these structures can easily be adopted
- There are good standards (RBAC und XACML)

Prerequisites

- You need a clear role model
- Roles must be mapped somehow in the user management system
  - Identity Management is quite helpful here
- Applications need to be able to consume such information
  - Role information can also be transported via federated Identity Management Systems (SAML, e.g. Shibboleth)
- You need an implementation

The RBAC standard

- ANSI INCITS 359-2004 says:
  - This standard describes RBAC features that have achieved acceptance in the commercial marketplace.
  - It includes a reference model and functional specifications for the RBAC features defined in the reference model.
  - It is intended for
    - software engineers and product development managers who design products incorporating access control features
    - managers and procurement officials who seek to acquire computer security products with features that provide access control capabilities in accordance with commonly known and understood terminology and functional specifications.

ANSI-Standard RBAC components

- The ANSI-Standard RBAC is divided into several functional components:
  - Core RBAC
    - Basic features every complient implementation must provide
  - Hierarchical RBAC (two types)
    - Optional role hierachies
  - Static Separation of Duty
    - Optional static exclusion of concurrency of single roles
  - Dynamic Separation of Duty
    - Optional dynamic exclusion of concurrency of single roles, i.e. at run-time

ANSI-Standard RBAC concepts

- object:
  - any system resource subject to access control, such as a file, printer, terminal, database record, etc.
- operation:
  - executable image of a program, which upon invocation executes some function for the user (e.g. read, write, execute, etc.).
- permissions:
  - an approval to perform an operation on one or more RBAC protected objects.

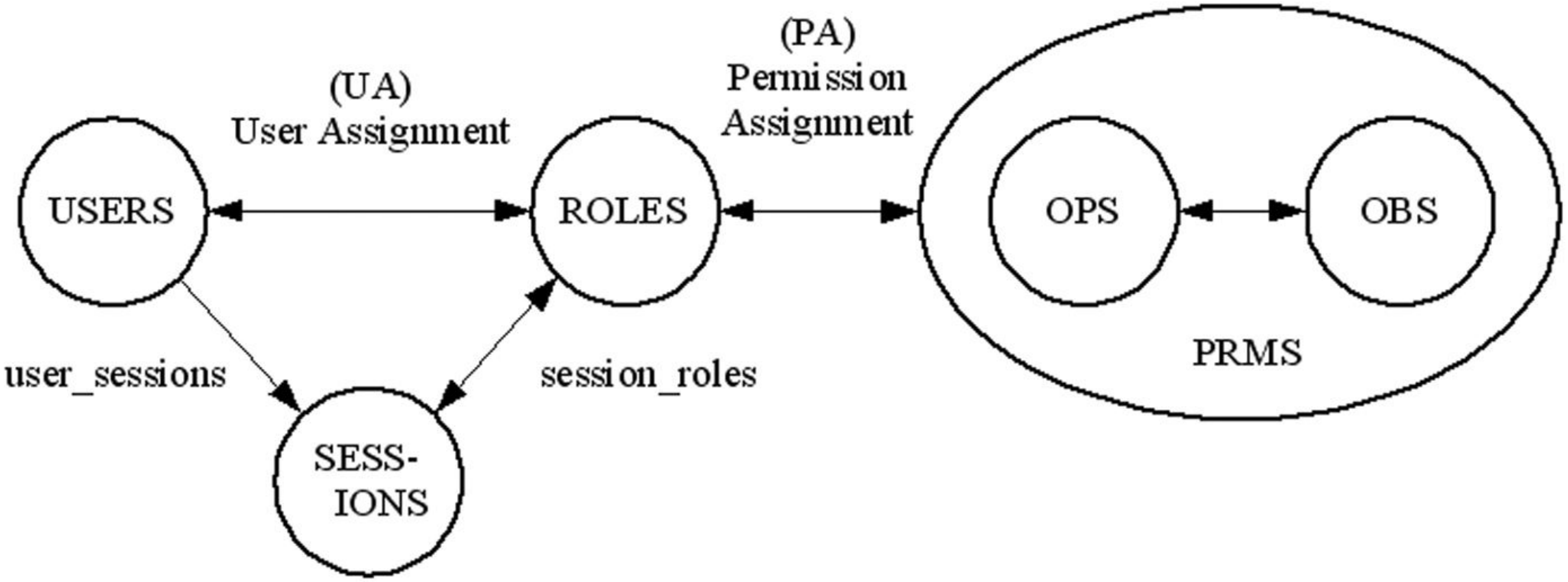
ANSI-Standard RBAC concepts

- role:
  - a job function within the context of an organization with some associated semantics regarding the authority and responsibility conferred on the user assigned to the role
- user:
  - a human being or any other agent in an IT system like machines, networks, or intelligent autonomous agents
- session:
  - A combination of a randomly (unique) ID, a user, a roleset and a lifetime

RBAC-Core

- Defines basic functionality, any implementation of the RBAC standard has to have. These are:
  - Creating and deleting users, roles and sessions
  - Creating and deleting permissions on resources
- Defines the function checkAccess, that can be used retrieve a decision for a object/operation combination
- Defines additional functionality
  - to change relationships between components e.g. add user to a role
  - to get information about single components of the system

RBAC-Core

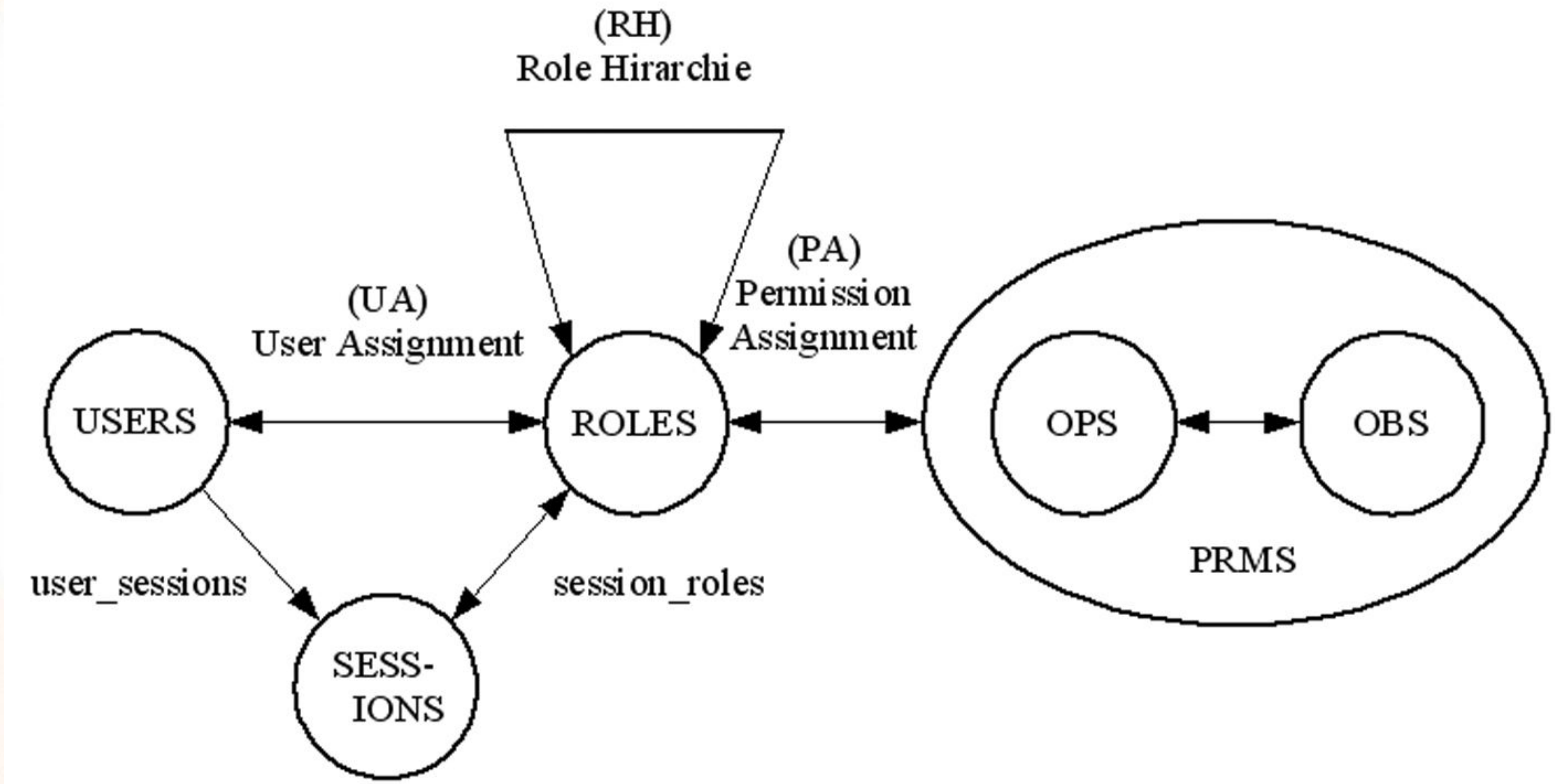

(UA)
User Assignment
(PA)
Permission
Assignment
USERS
ROLES
OPS
OBS
PRMS
user_sessions
session_roles
SESS-
IONS

Hierarchical RBAC

- Extends the basic functionality with role hierarchies defining two types:
  - Limited Role Hierarchy: roles are organized in a tree structure (single parent node, multiple child nodes)
  - General Role Hierarchy: roles are organized in free graphs (no limit to parent or child nodes)
- Some of the functions of RBAC-Core are adapted:
  - e.g. addActiveRole: now has to consider the hierarchy of roles when activating a session role
- In addition some new functionality is defined to change the role hierarchies

DAASI International

Hierarchical RBAC



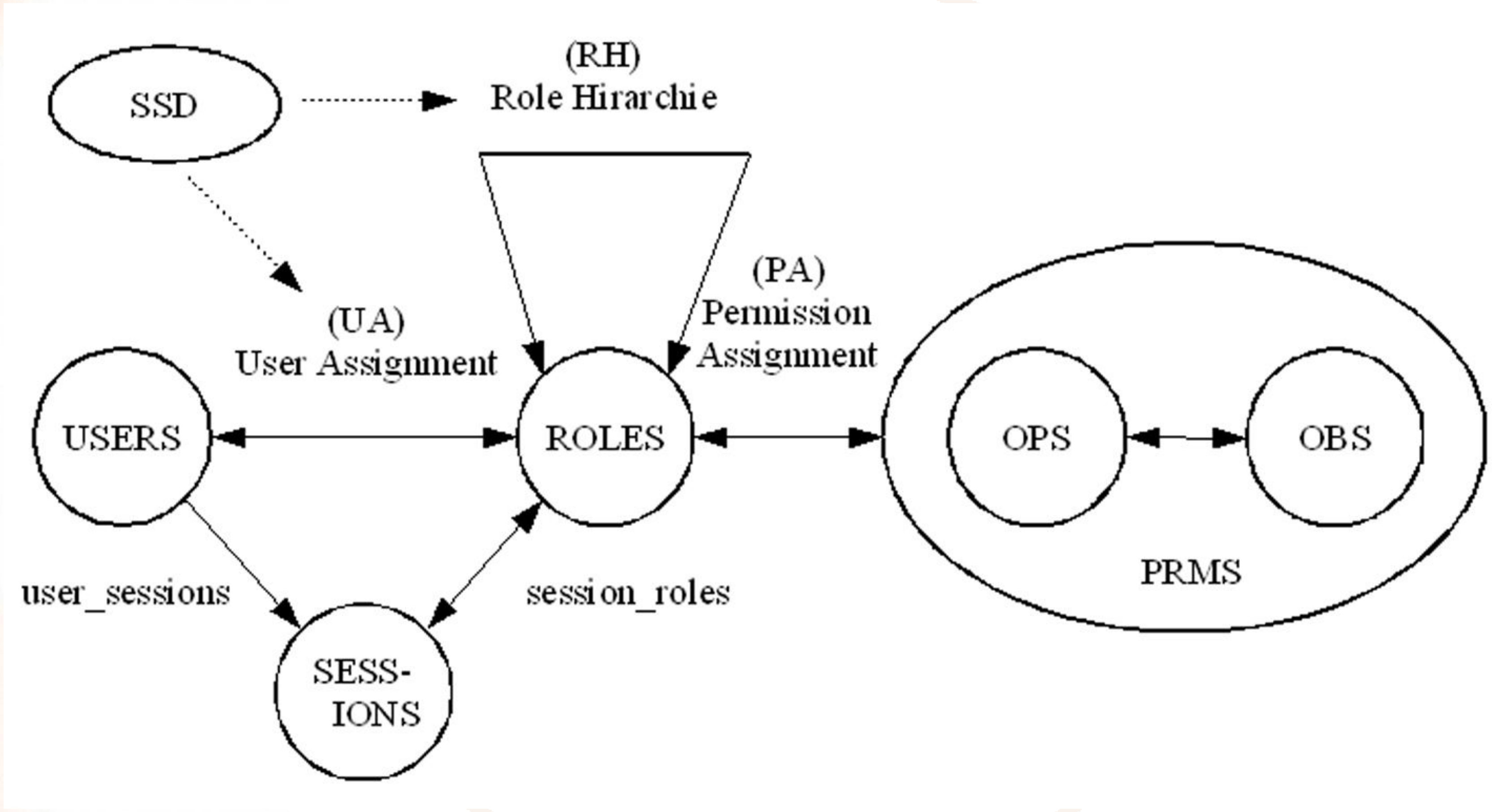
Separation of Duty

- Used to prevent users from getting or activating conflicting role combinations
- In cases of conflict of interests (e.g. applicant and allower)
- By using so called Sets roles can be defined that exclude one another
- There are two different types:
  - Static Separation of Duty (SSD)
  - Dynamic Separation of Duty (DSD)
- These two types can be used independently or in combination

Static Separation of Duty (SSD)

- SSD-Sets define two or more roles that cannot be assigned to the same user at any time
- These restrictions are checked each time a user is assigned to a role
- SSD relations define and place constraints on a user’s total permission space
- SSD relations may exist within hierarchical RBAC

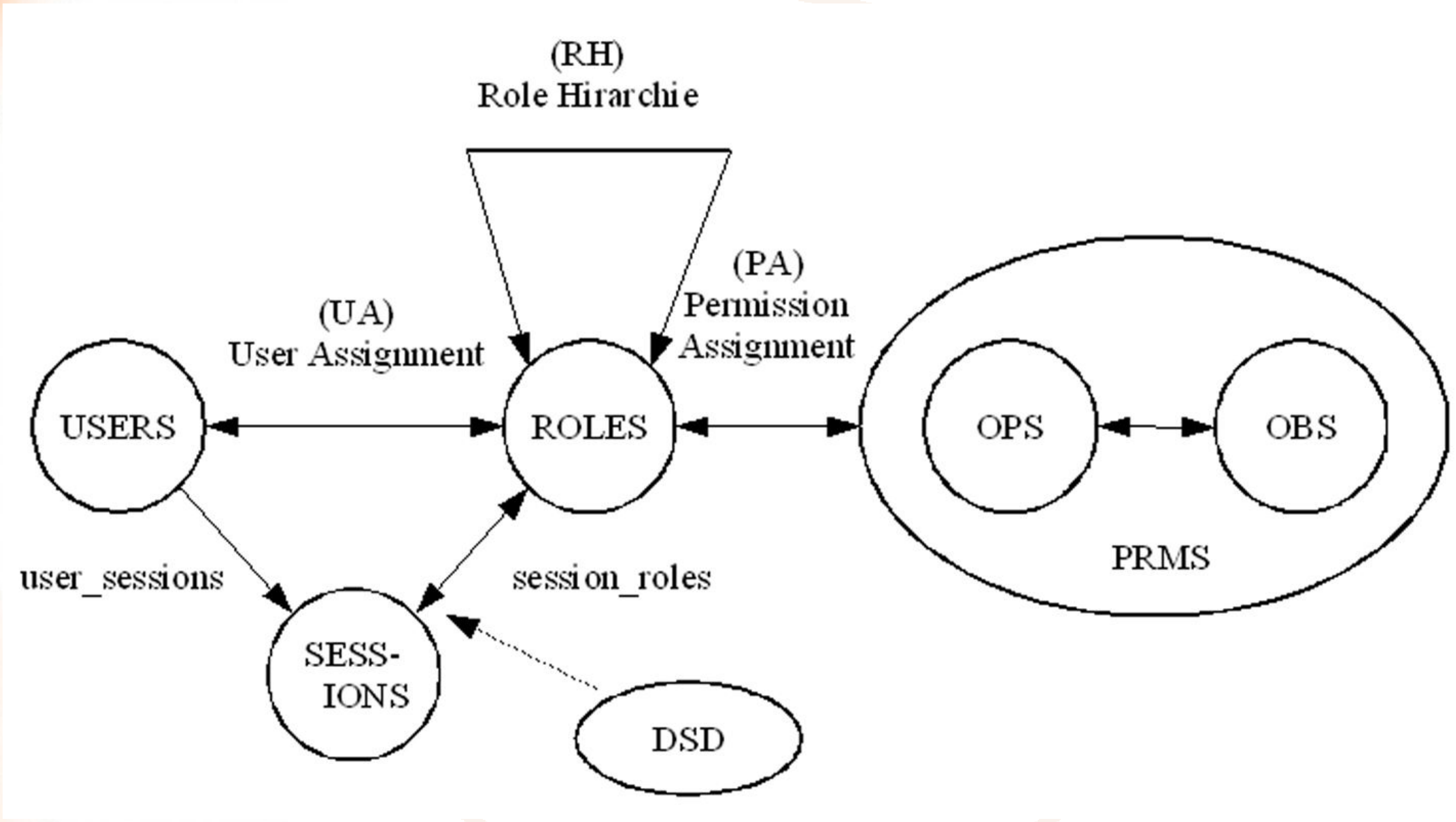
Static Separation of Duty (SSD)



Dynamic Separation of Duty (DSD)

- Restrictions are only checked when activating a role for a user's session
- Users are allowed to be assigned to roles that exclude on another but they are not allowed to activate them at the same time (i.e. in one session)
- Active roles are assigned to a user's session whereas a user can be assigned to more then these roles
- DSD properties provide extended support for the principle of least privilege in that each user has different levels of permission at different times, depending on the role being performed

Dynamic Separation of Duty (DSD)



Extensibility of RBAC

- The standard already defines a wide range of functions that provide many useful features for authorization tasks
- It can easily be extended as the standard itself is structured in basic functionality and additional extending modules
- An example for an extension could be e.g. „Multi-session Separation of Duties“ (David Chadwick, 2006):
  - An extension where a user is not only prevented from activating roles within the same session but across multiple active sessions

XACML: an interoperable standard

- Extended Access Control Markup Language
  - OASIS-Standard
  - There are XACML profiles for SAML and LDAP/DSML as well as for RBAC
- Access policies can be specified independent from applications
  - A policy can reference another policy
- Complex: like a programming language
  - Thus only slowly becoming accepted

XACML-Elemente

- PolicySet: Container for policies or other PolicySets
- Policies and PolicySets can be combined via algorithms
- Conditions are composed of subject, resource and action
- Policy Decision Point (PDP)
- Policy Enforcement Point (PEP)
- Target: collection of simple conditions
- Rule: Access control Rules
- Attributes

XACML-Request elements

- Subject
  - the object (person), that wants access to a resource
  - will be provided with attributes needed for the evaluation of policies (In the context of RBAC these attributes are roles)
- Resource
  - object, which is to be accessed
  - again provided with attributes needed for the evaluation of policies
- Action
  - Operation that is to be performed (e.g. read access to a resource)

XACML-Request-Protokoll

- SAML over SOAP can be used as request response protocol
- This way an XACML Policy Decision Point can be queried by a Web Service

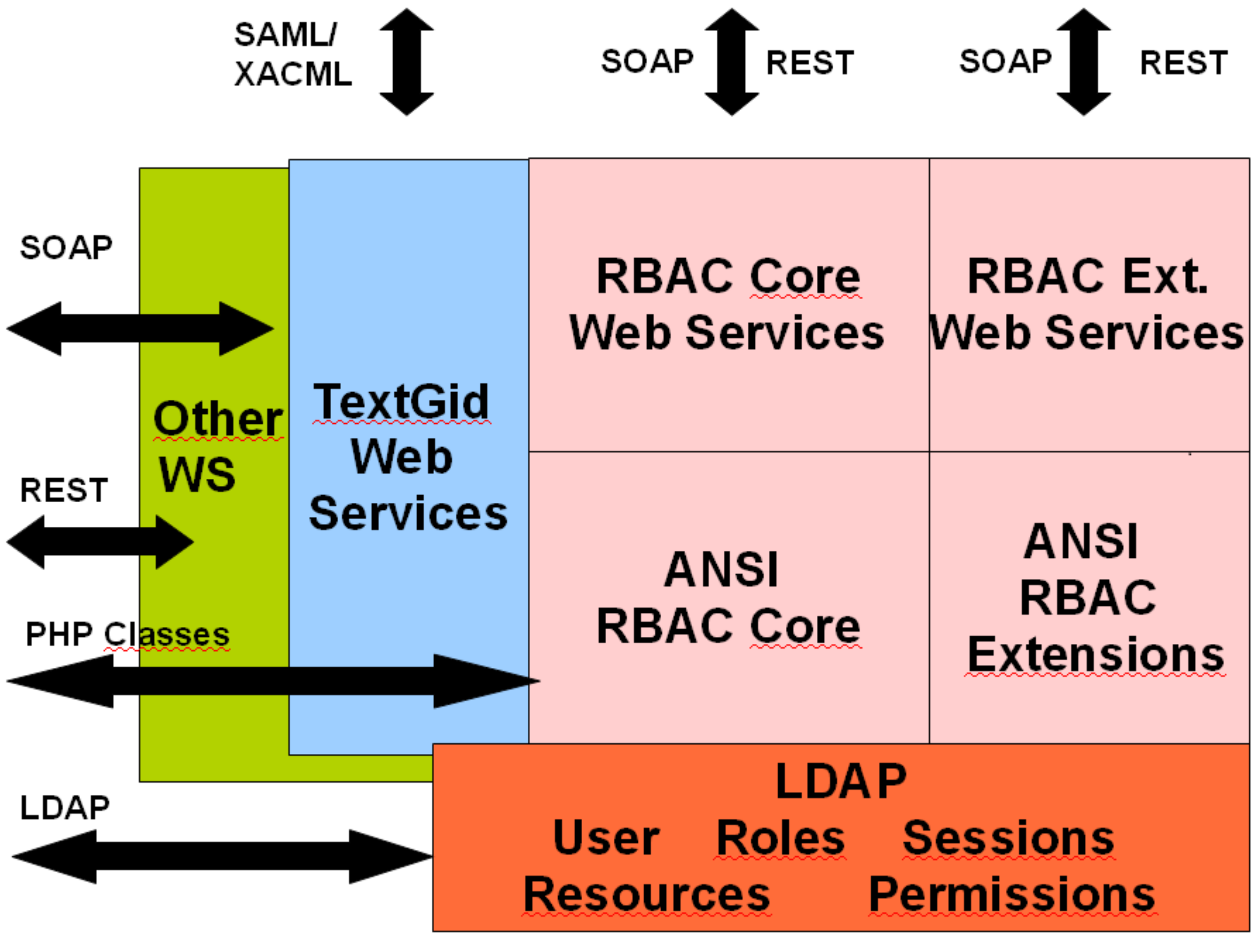
OpenRBAC

- OpenRBAC is an open source implementation of the standard
  - started as diploma thesis of Markus Widmer which was undertaken at DAASI International
  - it was used and extended by DAASI in the frame of several research projects on Grid Computing
  - It implements the complete standard except General Role Hierarchy (since Limited Role Hierarchies are mapped with the DIT)
  - thus all functions defined in the standard are implemented and accessible via SOAP web services, which can again be resources protected by OpenRBAC
- Documentation at http://www.openrbac.de. Most current sources are available at the TextGrid SVN

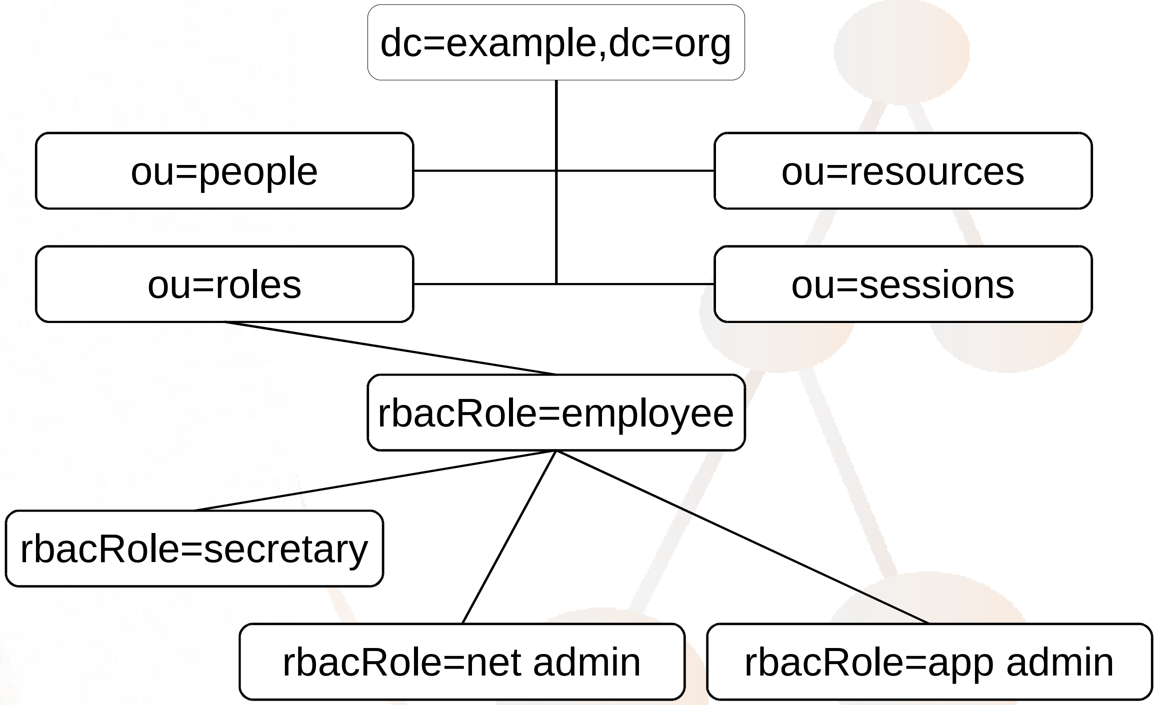
OpenRBAC Layer Model

- OpenRBAC is implemented on several distinct layers:
  - Core data base backend is an OpenLDAP server with respective DIT and Schema
  - All RBAC functions are implemented as methods of PHP classes, whereas the three components Core RBAC, Role Hierarchies and Separation of Duty are encapsulated
  - These PHP classes can be accessed via web service wrapper
  - Extended web services can also access the single RBAC methods
  - RBAC function checkAccess is also accessible via the mentioned XACML/SAML protocol

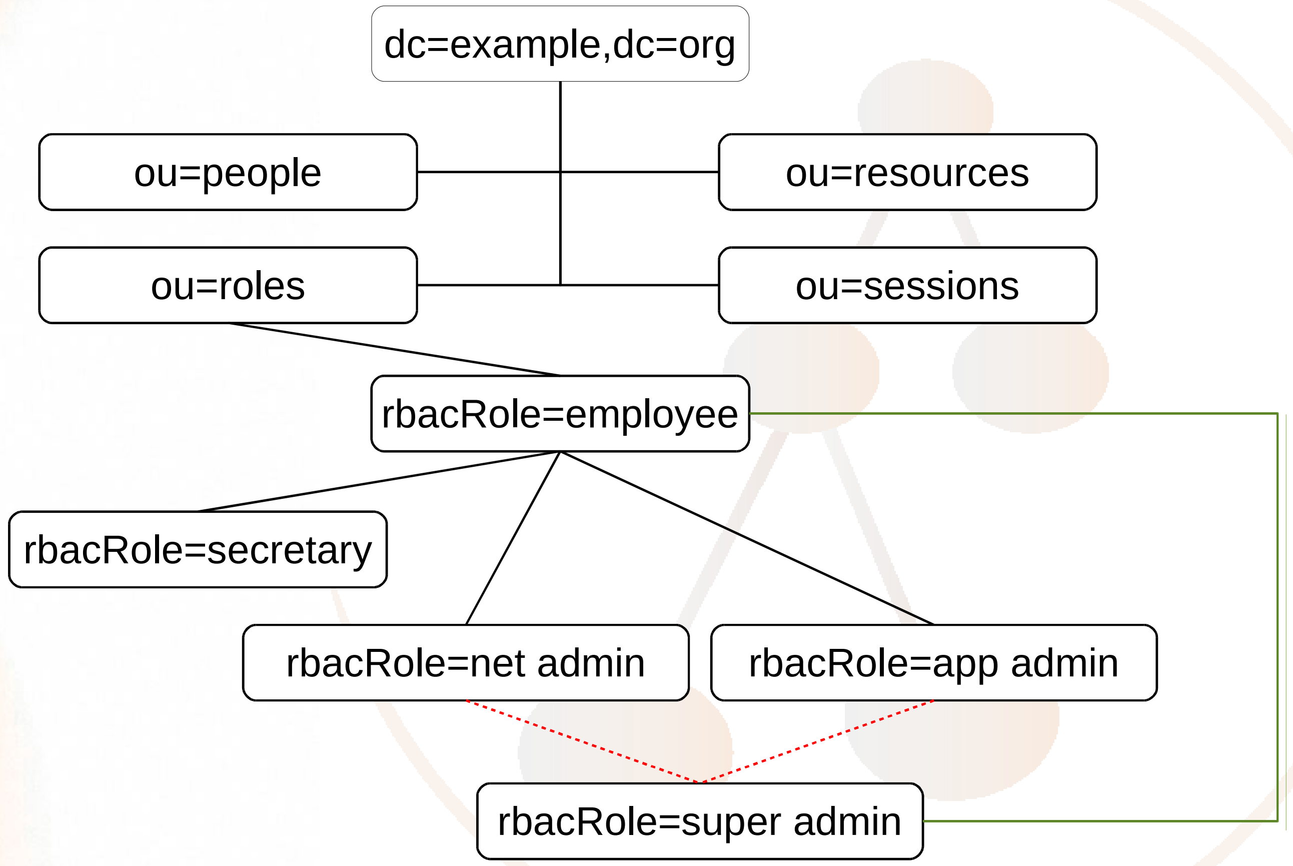
OpenRBAC Layer Model



OpenRBAC DIT and limited role hierarchy



OpenRBAC DIT and limited role hierarchy



OpenRBAC LDAP example

- **User**

```
dn: uid=test.user@example.com, ou=people,
dc=example,dc=org
objectClass: top
objectClass: person
objectClass: inetOrgPerson
cn: Test User
sn: User
givenName: Test
uid: test.user@example.com
```

OpenRBAC LDAP example

- Role

```
dn: rbacName=secretary, rbacName=employee, ou=roles,
dc=example,dc=org
objectClass: top
objectClass: rbacRole
rbacName: secretary
rbacPerformer: test.user@example.com
```

OpenRBAC LDAP example

- **Session**

```
dn: rbacName=74298fzzwjhb9, ou=sessions, dc=example,dc=org
objectClass: top
objectClass: rbacSession
rbacSessionCreationTimestamp: 20111011140000Z
rbacSessionCheckTimestamp: 20111011140113Z
rbacSessionUser: test.user@example.com
rbacSessionRole: rbacName=employee,ou=roles,dc=example,dc=org
rbacSessionRole: rbacName=secretary,rbacName=employee,
  ou=roles,dc=example,dc=org
```

OpenRBAC LDAP example

- **Resource**

```
dn: cn=Door 1, ou=resources, dc=example,dc=org
objectClass: top
objectClass: customResourceClass
objectClass: rbacResource
cn: Door 1
rbacOperation: open
rbacOperation: lock
rbacPermission:
rbacName=employee,ou=roles,dc=example,dc=org:=:open
rbacPermission: rbacName=secretary,rbacName=employee,ou=roles,
   dc=example,dc=org:=:lock
```

OpenRBAC LDAP example

- User test.user@example.com has roles
  - employee
  - secretary
- User has activated both roles in the session
  - by activating role secretary the role employee has been activated automatically
- May “open” and “lock” resource “Door1”

OpenRBAC used by the TextGrid project

- TextGrid develops a virtual research environment for the humanities, currently for:
  - philologists, linguists, musicologists, and art historians
- TextGrid software consists of two parts:
  - TextGridLab : GUI und web services based workbench
  - TextGridRep: Middleware for the management of information objects in a storage grid
- For authentication and authorization infrastructure OpenLDAP, OpenRBAC and Shibboleth (SAML) are deployed

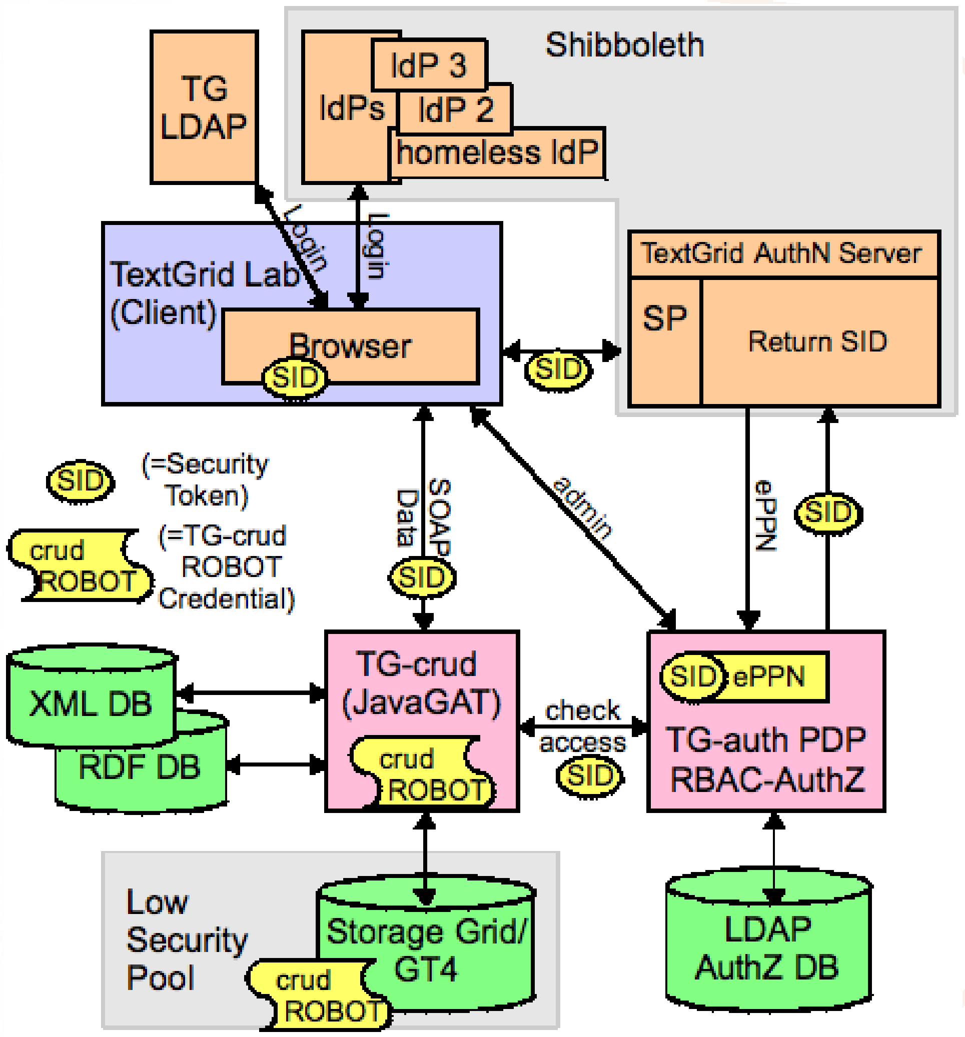
TextGridRep

- The TextGridRep (Repository) is a repository of humanities research data distributed in the grid that aims at long term accessibility
  - it can be accessed via well defined web services
- consists of:
  - TG-auth* for authentication and authorization (Shibbboleth and OpenRBAC)
  - TG-search: XML data base (eXist) for metadata and full texts, RDF data base (Sesame) for relations between the information objects
  - TG-crud Service (create/retrieve/update/delete): for managing the data in the grid
- It bridges LDAP/Shibboleth-AAI and PKI based Grid Security Infrastructure (GSI)

Three scenarios for integrating an external PDP into TextGrid

- 1.) Only TG-Crud contacts TGAuth (independent from the Globus grid middleware)
  - email based Verification of users stored in TextGrid LDAP server or coming from Shibboleth based authentication (for members of German higher edu organizations)
  - TG-crud authenticates to the Grid via a ROBOT certificate
- 2.) Mapping of the PDP policy on to POSIX ACLs
  - Creation of a Short Lived Credential PKI certificate upon a Shibboleth based authentication
  - Permissions are mapped on file level via POSIX ACLs and SLC is mapped to a unix user
- 3.) Direct Contacting the PDP via the grid middleware (XACML-Callout)
  - User permission are enforced by the Grid middleware

Szenario 1

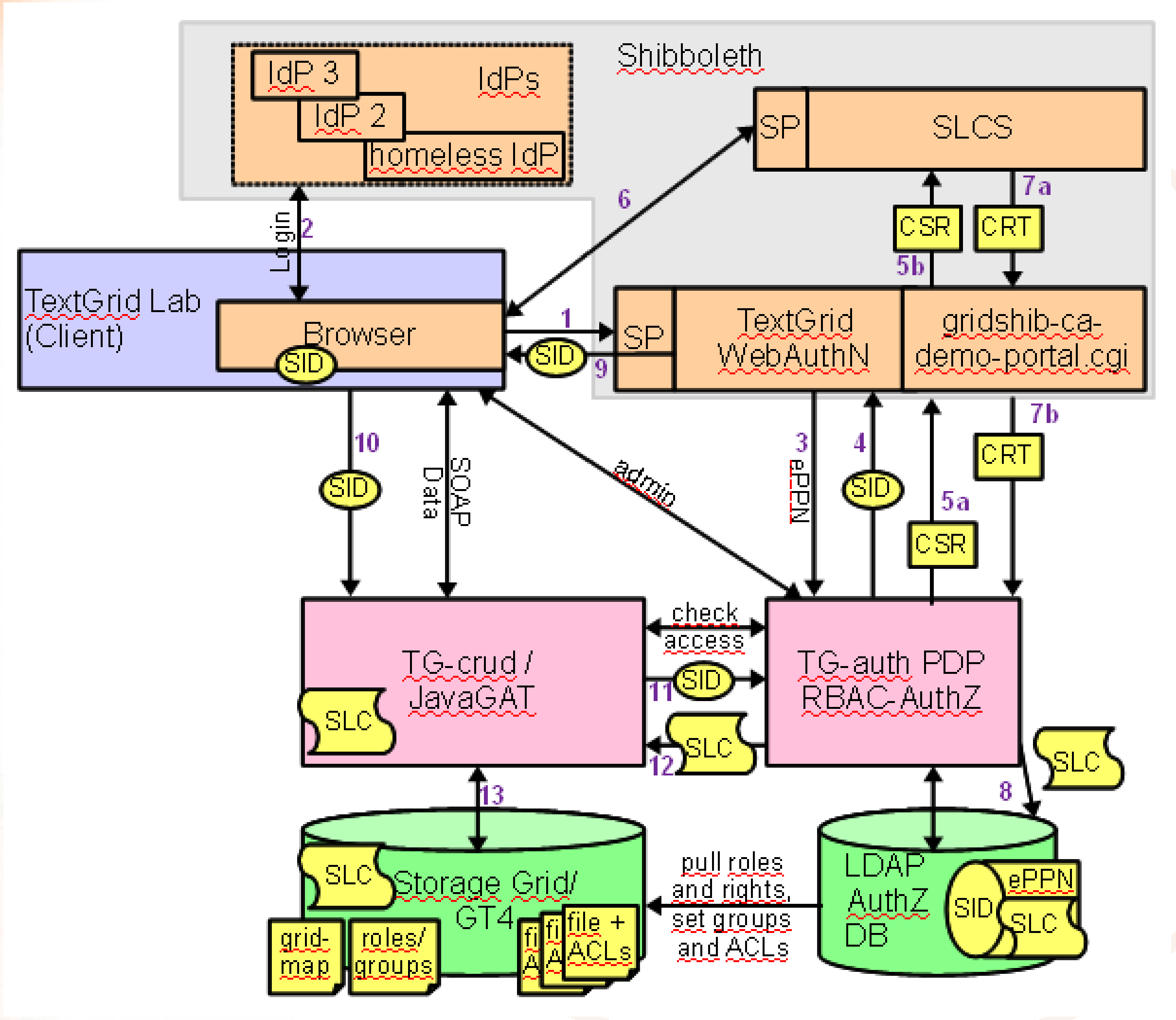


Shibboleth
TG LDAP
IdPs
IdP 3
IdP 2
homeless IdP
Login
Login
TextGrid Lab (Client)
Browser
SID
SID
TextGrid AuthN Server
SP
Return SID
SID (=Security Token)
crud ROBOT (=TG-crud ROBOT Credential)
SOAP Data
SID
admin
ePPN
SID
XML DB
RDF DB
TG-crud (JavaGAT)
crud ROBOT
check access
SID
SID ePPN
TG-auth PDP RBAC-AuthZ
Low Security Pool
Storage Grid/ GT4
crud ROBOT
LDAP AuthZ DB

Szenario 2

- User authenticates via Shibboleth (TG-auth*)
- TG-auth* generates a key pair and SLC certificate request (private key ist only stored in the RAM, not on HD)
- Portal redirects request to SLC service, that signs the SLC
- signed certificate is stored in TG-auth* (OpenLDAP)
- TG-crud retrieves SLC from TG-auth*
- data are stored as files in the home directory of the SLC user
- Access rights are mapped to the file system of the Grid node by TG-auth* (RBAC) via ACLs

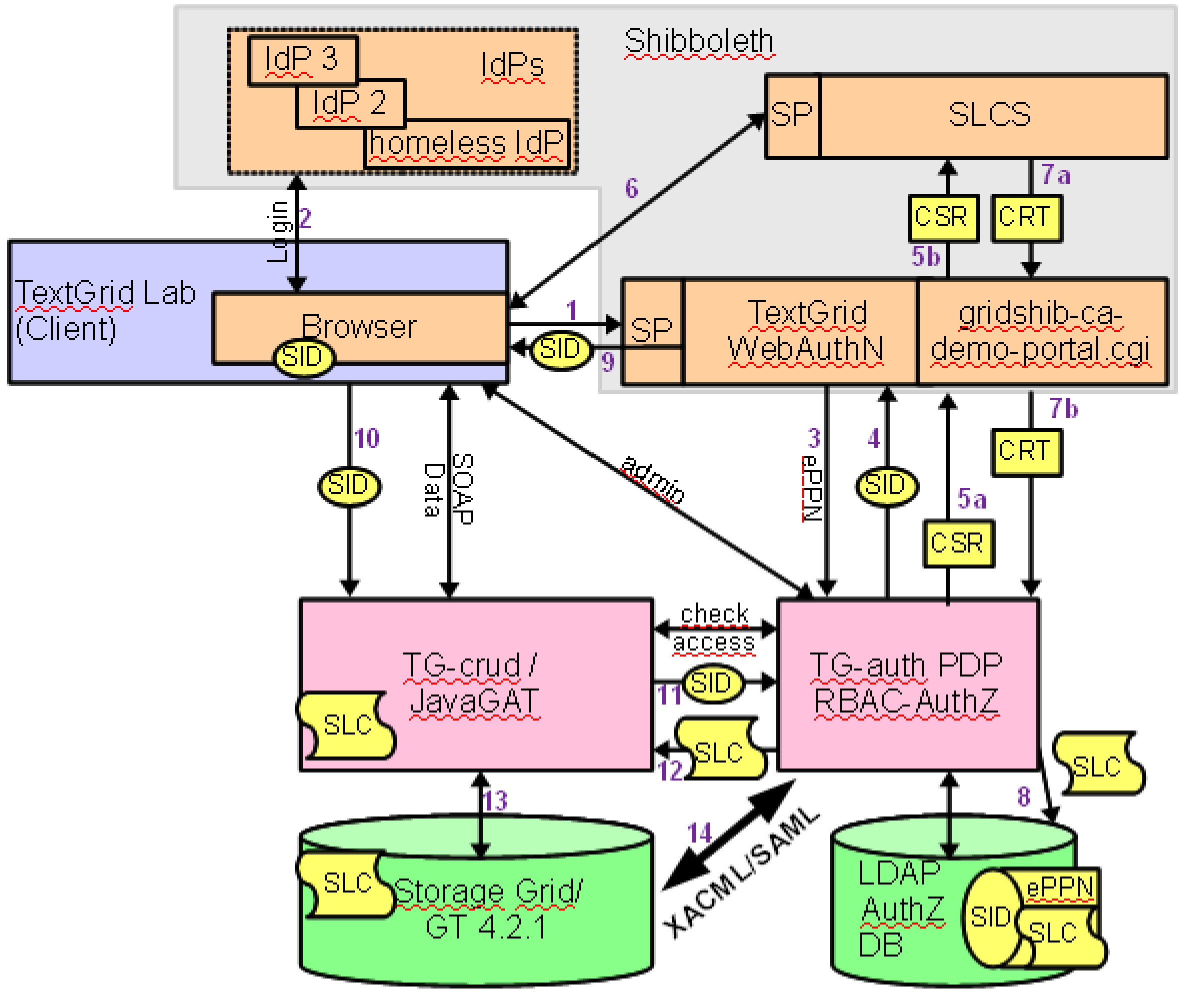
Szenario 2


Shibboleth
IdP 3
IdPs
IdP 2
homeless IdP
SP
SLCS
7a
CSR
CRT
5b
6
Login
2
TextGrid Lab
(Client)
Browser
SID
1
SID
9
SP
TextGrid
WebAuthN
gridshib-ca-
demo-portal.cgi
10
SID
SOAP
Data
admin
3
ePPN
4
SID
5a
CSR
7b
CRT
TG-crud /
JavaGAT
SLC
check
access
11
SID
12
SLC
TG-auth PDP
RBAC-AuthZ
SLC
8
13
SLC
Storage Grid/
GT4
grid-
map
roles/
groups
file +
ACLs
pull roles
and rights,
set groups
and ACLs
LDAP
AuthZ
DB
SID
ePPN
SLC

Szenario 3 - XACML-SAML

- SLCs just like scenario 2
- Data again stored in below the home directory of the SLC user
- Access rights are queried directly by Globus as PEP from TG-auth* as PDP via the XACML SAML request response protocol, with the following parameters:
  - Subject-DN from the certificate
  - Name of the resource (file)
  - operation (read, write, ...)
- On file level Globus has all rights on all resources via group membership

Szenario 3 - XACML-SAML



Discussion

- OpenRBAC is a mere backend infrastructure
  - that can flexibly be deployed in very different contexts
- It is transparent for OpenRBAC:
  - whether roles are managed centrally or locally
    - OpenRBAC methods are resources protected by OpenRBAC
    - „central PDP“ ist central for an application but could as well be managed locally
  - whether two-person integrity is introduced
    - This has to be handled in the front end

Why OpenLDAP as backend?

- Parts of the information needed is very often already stored in a LDAP server (user objects)
  - Existing user and or resource data can directly be used or extended to fit the RBAC needs
  - The data stored in a directory can easily be (re)used by other applications
- Different information objects are organised in different subtrees (ou=Roles, ou=Resources, ou=Sessions, etc.) and thus can easily be distributed on several LDAP servers, e.g.:
  - User on the authentication server and all other data on a dedicated second LDAP server

Why OpenLDAP as backend?

- (Open)LDAP can give fast replies to access queries with the data model chosen
  - CheckAccess is implementable by a single LDAP filter
  - Through Howard's mesurements we all know how fast OpenLDAP can be on a decent hardware: It can handle over 60,000 search requests per second even on large data bases of one million entries

Why OpenLDAP as backend?

- A resource filter functionality can be easily implemented constructing a complex filter:
  - (| (& (resID=x1) (perm=role1:=:read))
    (& (resID=x2) (perm=role2:=:read))
    ...)
- Only resources with correct permissions are returned

Future work:
Using ACLs to reduce code functionality

- Change data structure to have permission objects instead of storing permissions within resource objects:

```
rbacName: perm1
rbacRole: secretary
rbacOperation: lock
rbacResource: Door1
```

- Add ACL

```
access to dn.one=”ou=permissions,dc=example,dc=org”
  by set=”this/rbacRole & user/rbacRole” read
```

- Do a authzTo session object
- Search for (&(rbacResource=Door1)(rbacOperation=lock))
  - Number of results > 0 → access granted

Using ACLs to reduce code functionality

- Pro:
  - No need to extract session roles and create a complex filter containing all these roles anymore
  - Changing datastructure to have permission objects would allow to add additional constraints for each permission
- Cons:
  - Data structure needs to be changed (without backwards compatibility)
  - There are as many permissions as resource objects or very large permission objects

Disadvantages using LDAP as backend

- Storing role relationships other than tree structures is difficult and there will not be any benefit any more to use the DIT hierarchy
- Some actions have to do multiple LDAP queries when relational databases can often do this in one single query, e.g. the action „grant a permission“ has to:
  - Search the resource (and get the possible actions on the resource)
  - Search the user corresponding to the session (RBAC operates on sessions, not on users)
  - Search the roles the user is assigned to
  - Set the permission by modifying the resource

Conclusion

- The current version of OpenRBAC can profit from the hierarchical structure of LDAP a lot, since it does not support complex (=non-tree) role hierarchies.
    - Such complex role structures will not be necessary in most cases
    - One case that could benefit from complex role hierarchy:
        - Super admin who has the admin rights of all department admins
        - Such a super admin role would have to be specified outside of the hierarchy and will have to be included into every resource entry, that contain the permissions
- checkAccess can be implemented by one LDAP filter (after once having found out the active roles of a user)
- The fast read access of LDAP makes such policy decision request very fast

Conclusion

- By using LDAP as backend openRBAC can easily be distributed and thus is highly scalable and fit for high availability scenarios
- The design of OpenRBAC helped integrating a PDP into a variety of deployment scenarios
- It can very well adapt to new requirements via its extension mechanism
- By using ACLs in a next version, it will be even more powerful, easier to use and most probably faster in response
- It once again shows that you can do more with LDAP than just storing user information and passwords

Thanks a lot for your attention!

- Questions?
- DAASI International GmbH
  - www.daasi.de
  - Info@daasi.de