EUROPEAN MIDDLEWARE INITIATIVE

How Argus can simplify your life

Andrea Ceccanti (INFN)
On behalf of the Argus PT

What is authorization?

Can user X
perform action Y
on resource Z
?

Authorization examples

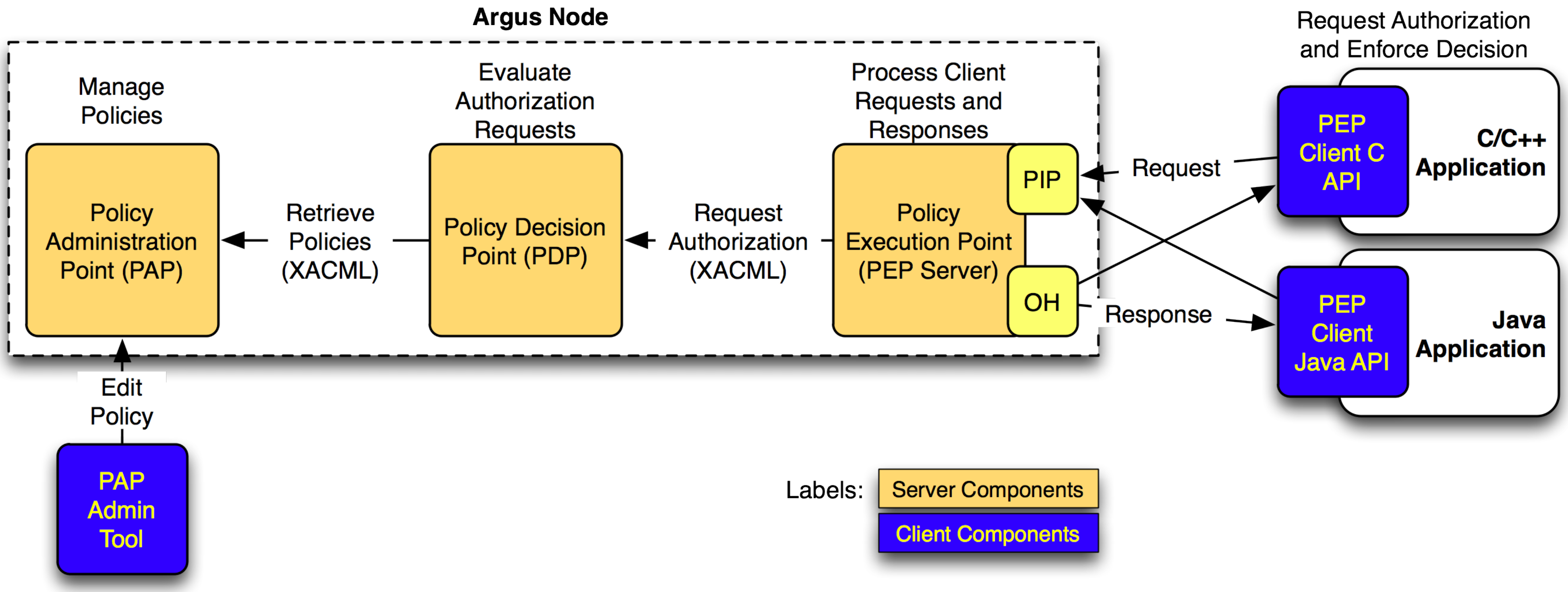
- Can user X
  - execute on this Worker node?
  - submit a job on this CREAM CE?
  - access this storage area?
  - submit a job on this WMS instance?

Motivations for Argus

- Each Grid service has its own authorization mechanism
  - Administrators need to know them all
  - Authorization rules at a site are difficoult to understand and manage
- No global banning mechanism
  - Urgent ban of malicious users cannot be easily and timely implemented on distributed sites
- Authorization policies are static
  - Hard to change policies without reconfiguring services
- Monitoring AuthZ decisions is hard

Argus

- A generic authorization system
  - Built on top of an XACML policy engine
  - renders consistent authorization decisions based on XACML policies


Argus Node
Manage Policies
Policy Administration Point (PAP)
Retrieve Policies (XACML)
Evaluate Authorization Requests
Policy Decision Point (PDP)
Request Authorization (XACML)
Process Client Requests and Responses
Policy Execution Point (PEP Server)
PIP
OH
Request
Response
Request Authorization and Enforce Decision
PEP Client C API
C/C++ Application
PEP Client Java API
Java Application
Edit Policy
PAP Admin Tool
Labels:
Server Components
Client Components

Argus components

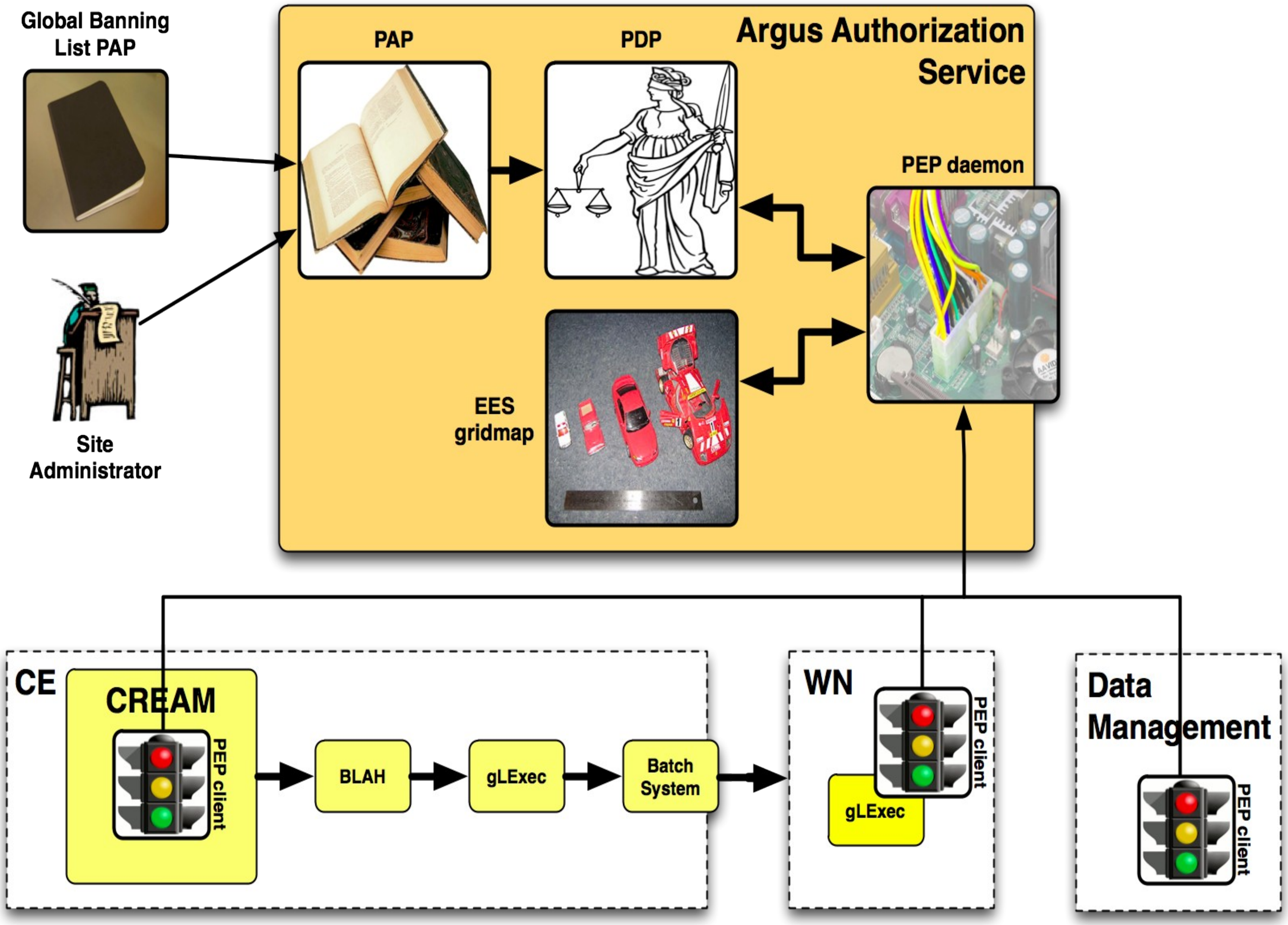
- Policy Administration Point (PAP)
  - Provides administrators with the tools to author policies
  - Stores and manages authored XACML policies
  - Provides managed authorization policies to other authorization service components (other PAPs or PDPs)

- Policy Decision Point (PDP)
  - Policy evaluation engine
  - Receives authorization requests from the PEP
  - Evaluates requests against policies fetched from the PAP
  - Renders the authorization decision

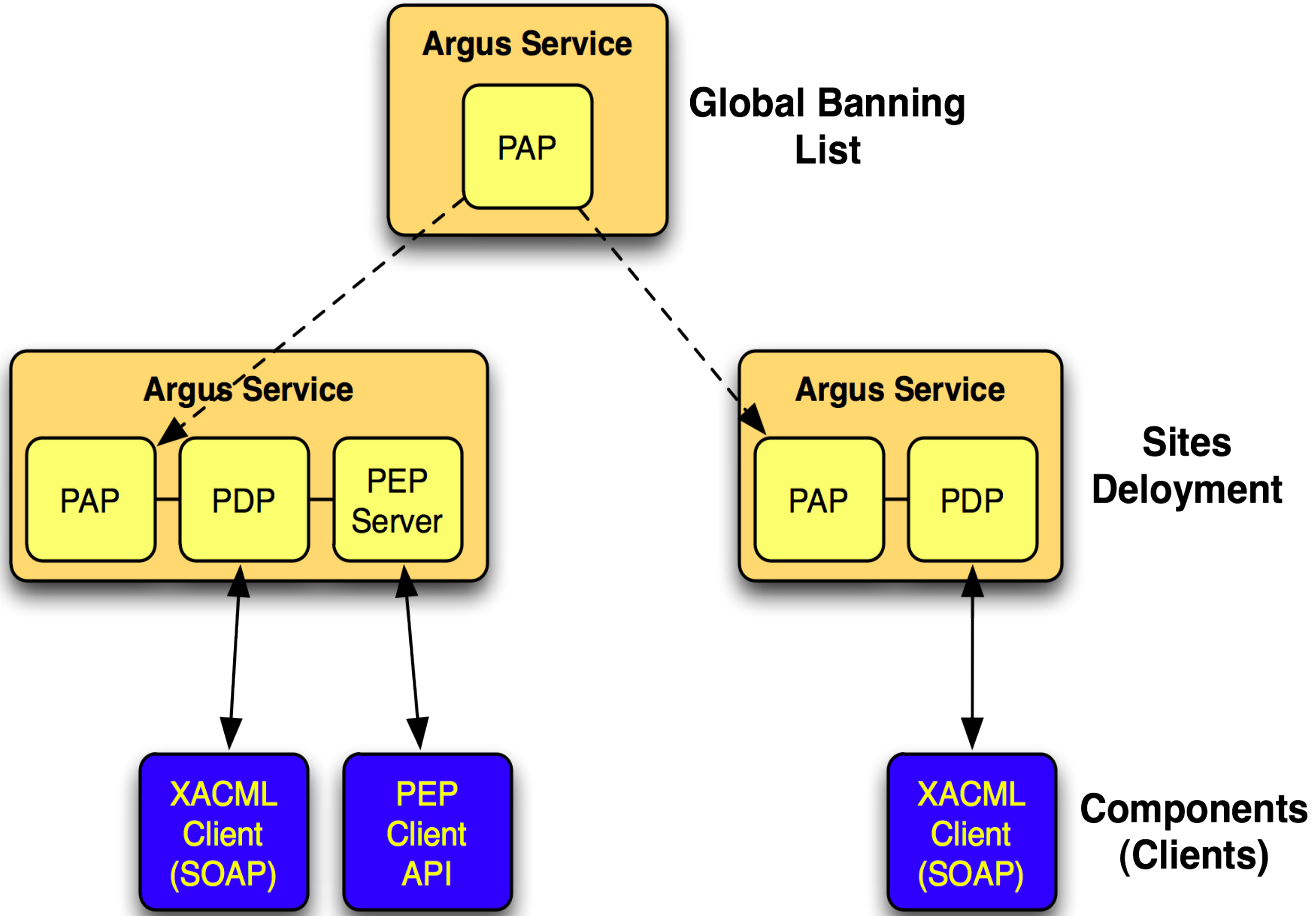
Argus components (cont.)

- Policy Enforcement Point (PEP)
  - Client/Server architecture
  - Lightweight PEP client libraries (C and Java)
  - PEP server receives the authorization requests from PEP clients
    - Transform lightweight internal request in XACML request
    - Applies a configurable set of filters (PIPs) to the incoming request
    - Asks the PDP to render an authorization decision
    - Applies Obligation Handler to determine user mapping, if requested by the PDP

Argus service deployment



Hierarchical policy distribution



Argus policies

Argus is designed to answer the question:

Can user X perform action Y on resource Z?

Argus policies contain rules that state which actions can be performed on which resources by which users.

Argus uses XACML v.2.0 as the policy language.

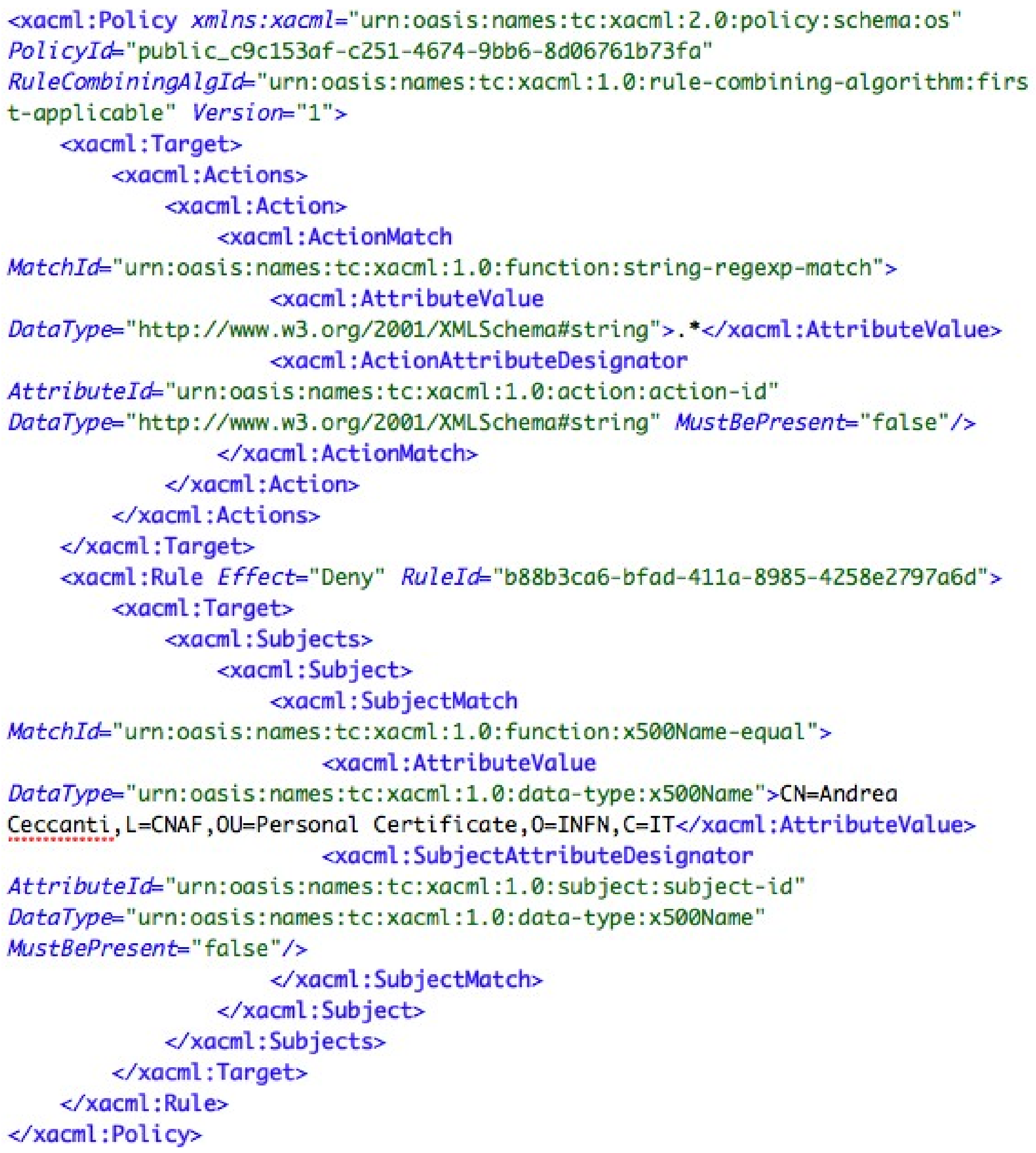
- however, XACML is hard to read and write, so we developed a **Simplified Policy Language (SPL)** to suit our needs

The banning use case

User X should not be allowed to do anything on any resource

(for this example X will be my certificate subject)

The XACML banning policy



```
<xacml:Policy xmlns:xacml="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
PolicyId="public_c9c153af-c251-4674-9bb6-8d06761b73fa"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:firs
t-applicable" Version="1">
    <xacml:Target>
        <xacml:Actions>
            <xacml:Action>
                <xacml:ActionMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
                    <xacml:AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">.*</xacml:AttributeValue>
                    <xacml:ActionAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
                </xacml:ActionMatch>
            </xacml:Action>
        </xacml:Actions>
    </xacml:Target>
    <xacml:Rule Effect="Deny" RuleId="b88b3ca6-bfad-411a-8985-4258e2797a6d">
        <xacml:Target>
            <xacml:Subjects>
                <xacml:Subject>
                    <xacml:SubjectMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:x500Name-equal">
                        <xacml:AttributeValue
DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name">CN=Andrea
Ceccanti,L=CNAF,OU=Personal Certificate,O=INFN,C=IT</xacml:AttributeValue>
                        <xacml:SubjectAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name"
MustBePresent="false"/>
                    </xacml:SubjectMatch>
                </xacml:Subject>
            </xacml:Subjects>
        </xacml:Target>
    </xacml:Rule>
</xacml:Policy>
```

...?

The SPL policy

```
resource ".*" {

    action ".*" {

        rule deny {

            subject="CN=Andrea Ceccanti,L=CNAF,OU=Personal Certificate,O=INFN,C=IT"

        }

    }
}
```

Identifying Actions and Resources

- Actions and resources are identified by unique “names” that are assigned to them
  - Tipycally URIs, but any string will work
- Resource ID example:
  - http://authz.cnaf.infn.it/resource/cream-ce
- Action ID example:
  - http://authz.cnaf.infn.it/action/submit-job

Identifying subjects

A Subject in a policy can be identified via the following attributes:

- **subject**, an X509 certificate subject

subject="CN=Andrea Ceccanti,L=CNAF,OU=Personal Certificate,O=INFN,C=IT"

- **ca**, the CA certificate subject

ca="CN=INFN CA,O=INFN,C=IT"

- **vo**, the name of the Virtual Organization

vo="cms"

- **fqan**, a VOMS fully qualified attribute name

fqan="/atlas/production"

- **pfqan**, the VOMS primary FQAN

pfqan="/atlas/Role=pilot"

SPL Syntax

```
resource <value> {

    action <value> {

        rule <permit|deny> {

        <attributeId> = <attributeValue>

        }
        ...
    }
    ...
}
...
```

Order matters!

Examples

We have two CEs at our site, ce_1 and ce_2. We want to authorize Andrea Ceccanti to contact one but not the other.

```
resource “ce_1”{
action “.*” {
        rule permit {
            subject = “CN=Andrea Ceccanti,L=CNAF,OU=Personal Certificate,O=INFN,C=IT”
        }
}
}

resource “ce_2”{
action “.*” {
        rule deny{
            subject = “CN=Andrea Ceccanti,L=CNAF,OU=Personal Certificate,O=INFN,C=IT”
        }
}
}
```

Examples

We want to ban users from VO ‘test_vo’ from ce_1 but not those who have a certificate signed by INFN CA.

```
resource “ce_1”{
action “.*” {

        rule permit {
            vo = “test_vo”
            ca = “CN=INFN CA,O=INFN,C=IT”
        }

        rule deny { vo = “test_vo” }
}
}
```

The pap-admin tool

- List currently active policies:
  - pap-admin list-policies
- Import policies expressed in SPL from a file:
  - pap-admin add-policies-from-file policies.txt
- Ban/Unban users:
  - pap-admin ban vo atlas
  - pap-admin unban vo atlas
- Add a generic permit/deny policies
  - pap-admin add-policy --resource “ce_1” --action “.*” \
    permit fqan=”/atlas/production”

Current Argus release

- Argus 1.3
  - First EMI release
  - Backwards compatible with gLite 3.2 client libraries
  - Support for LFC/DPM banning engine
  - Bug fixes
- https://savannah.cern.ch/task/?18586

So why Argus simplifies my life?

- A single authorization point for many grid services
- A simple, flexible and powerful language to express authorization policies
- A simple tool to manage policies
- A policy distribution mechanism that allow to import policies from remote sites while keeping full authorization control on local resources

Documentation

- General documentation
  https://twiki.cern.ch/twiki/bin/view/EGEE/AuthorizationFramework
- Service Reference Card
  https://twiki.cern.ch/twiki/bin/view/EMI/ArgusSRC
- PAP admin CLI
- https://twiki.cern.ch/twiki/bin/view/EGEE/AuthZPAPCLI
- Simplified Policy Language
  https://twiki.cern.ch/twiki/bin/view/EGEE/SimplifiedPolicyLanguage

EUROPEAN MIDDLEWARE INITIATIVE

Thanks!

EMI is partially funded by the European Commission under Grant Agreement INFSO-RI-261611