Federated Authority

HOOMAN BEHNEJAD

Agenda

- Problem
- Solutions
- Requirements
- Limitation
- Comparison

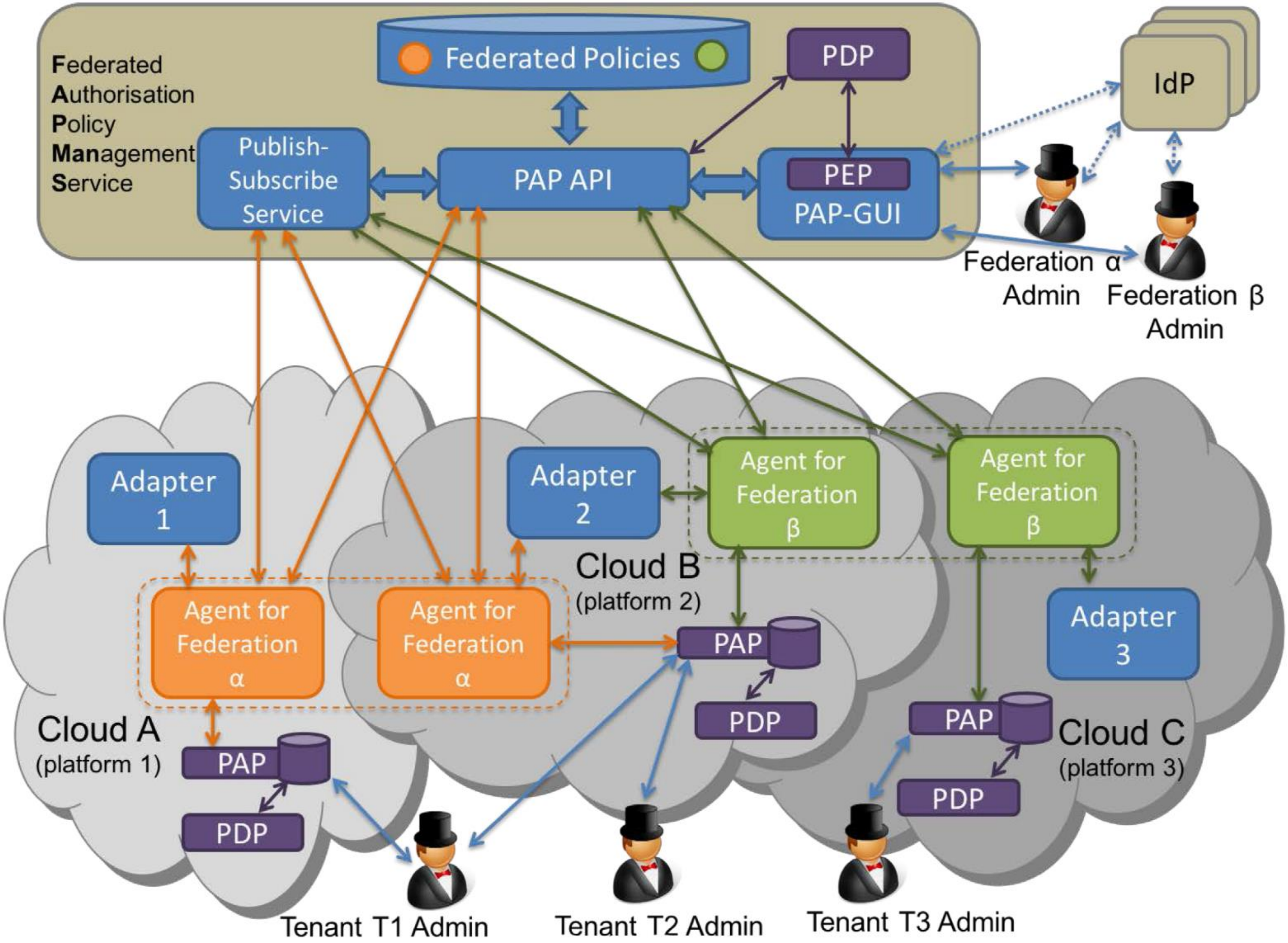
Problem

- Today each cloud provider has its own proprietary authorization system, containing different access control rules and models
- Even with federated authentication, a user may need different credentials to access different clouds
- If you have a multi-cloud environment, or a federation of heterogeneous cloud providers, how can you have a homogeneous authorization policy that applies equally for all users across all clouds?

Solution

- An Authorization Policy Federation – a group of heterogeneous cloud providers that agree to cooperate together in the management of their authorization policies.
- Has a federation wide Policy Administration Point, that stores conceptual abstract authorization policies using a cloud-independent ontology.
- Have mapping engines (adapters) that convert the abstract policies into cloud dependent policies (and vice versa) so that they can be enforced using the existing cloud authorization mechanisms.
- Have a publish-subscribe infrastructure that keeps the abstract and cloud dependent policies synchronized

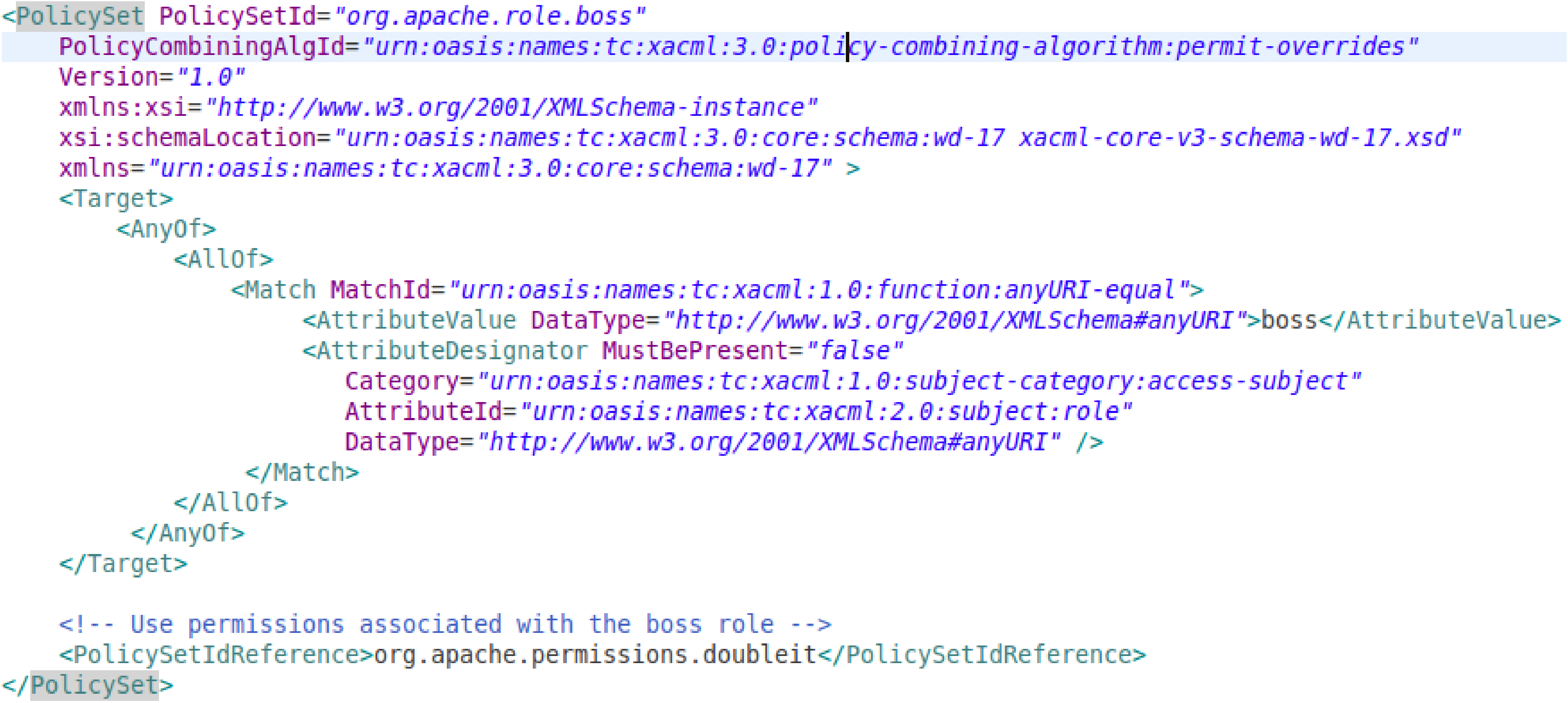
FAPManS architecture for policy administration



Abstract Policies

- standard format such as XACML
  - Pros: Standard, supports all AC models and policies
  - Cons: Verbose, Difficult to read/understand, slow to process, has an excess of features
- abstract format like Disjunctive Normal Form (DNF)
  - Pros: Easy to understand and represent in RDBMS, fast to process, can represent any set of policy conditions
  - Cons: Cannot support rich AC features such as obligations, different conflict resolution rules etc.

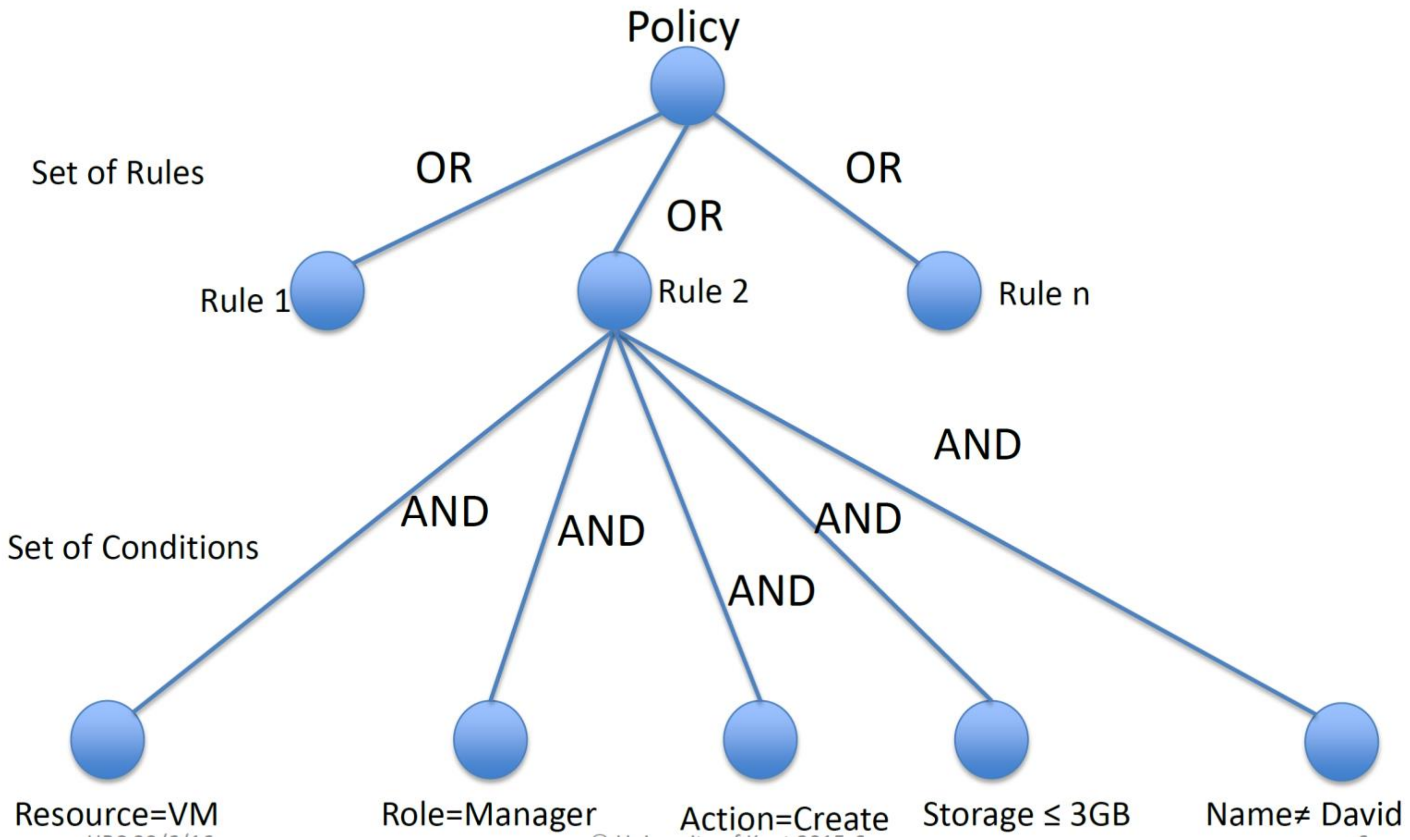
XACML Sample



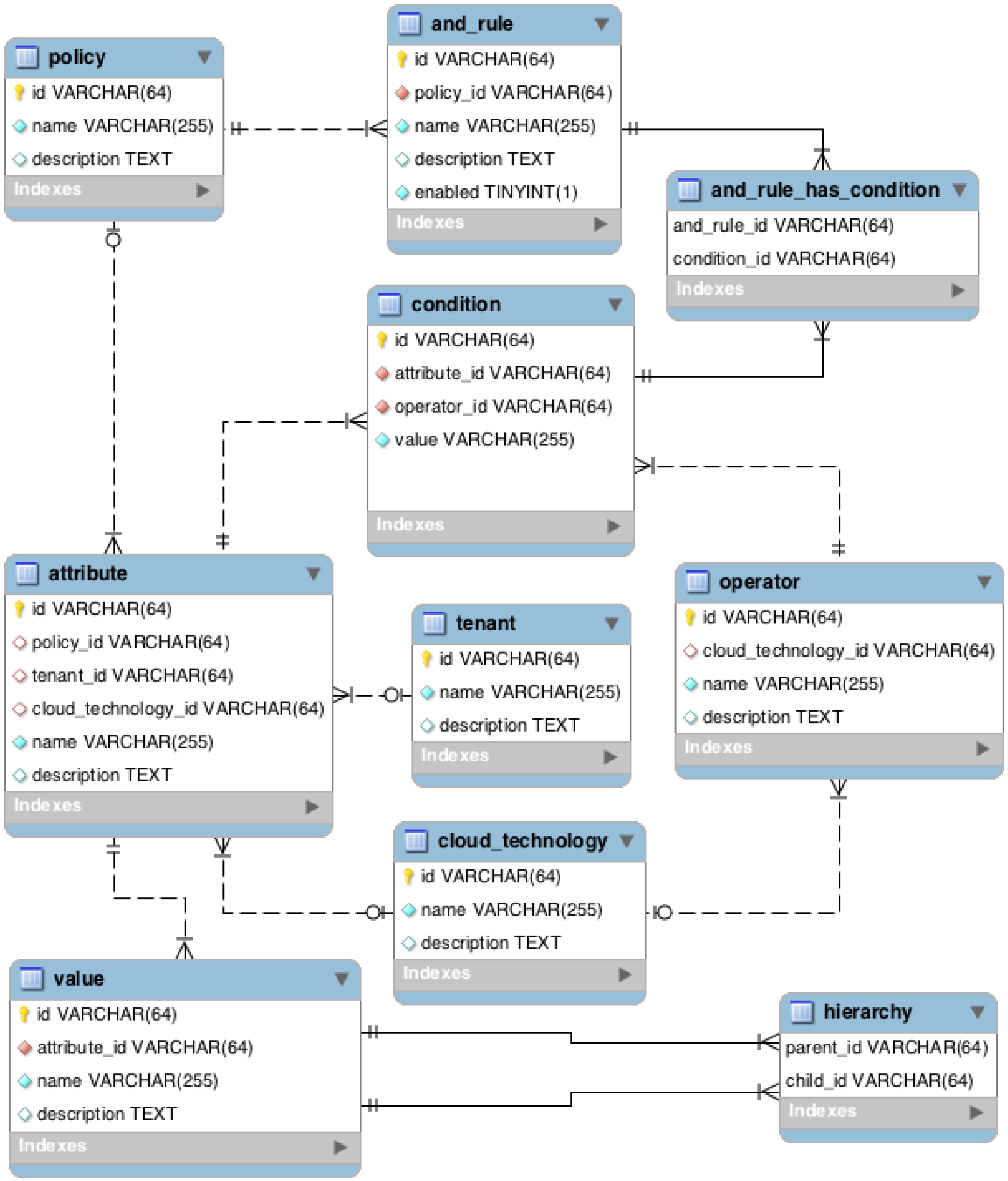
```
<PolicySet PolicySetId="org.apache.role.boss"
    PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-overrides"
    Version="1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17 xacml-core-v3-schema-wd-17.xsd"
    xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" >
    <Target>
        <AnyOf>
            <AllOf>
                <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
                     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">boss</AttributeValue>
                     <AttributeDesignator MustBePresent="false"
                        Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
                        AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
                        DataType="http://www.w3.org/2001/XMLSchema#anyURI" />
                 </Match>
            </AllOf>
         </AnyOf>
    </Target>

    <!-- Use permissions associated with the boss role -->
    <PolicySetIdReference>org.apache.permissions.doubleit</PolicySetIdReference>
</PolicySet>
```

DNF Sample


Policy
Set of Rules
OR
OR
OR
Rule 1
Rule 2
Rule n
AND
AND
AND
AND
AND
Set of Conditions
Resource=VM
Role=Manager
Action=Create
Storage ≤ 3GB
Name≠ David

Database Policy Schema



policy
id VARCHAR(64)
name VARCHAR(255)
description TEXT
Indexes

and_rule
id VARCHAR(64)
policy_id VARCHAR(64)
name VARCHAR(255)
description TEXT
enabled TINYINT(1)
Indexes

and_rule_has_condition
and_rule_id VARCHAR(64)
condition_id VARCHAR(64)
Indexes

condition
id VARCHAR(64)
attribute_id VARCHAR(64)
operator_id VARCHAR(64)
value VARCHAR(255)
Indexes

attribute
id VARCHAR(64)
policy_id VARCHAR(64)
tenant_id VARCHAR(64)
cloud_technology_id VARCHAR(64)
name VARCHAR(255)
description TEXT
Indexes

tenant
id VARCHAR(64)
name VARCHAR(255)
description TEXT
Indexes

operator
id VARCHAR(64)
cloud_technology_id VARCHAR(64)
name VARCHAR(255)
description TEXT
Indexes

cloud_technology
id VARCHAR(64)
name VARCHAR(255)
description TEXT
Indexes

value
id VARCHAR(64)
attribute_id VARCHAR(64)
name VARCHAR(255)
description TEXT
Indexes

hierarchy
parent_id VARCHAR(64)
child_id VARCHAR(64)
Indexes

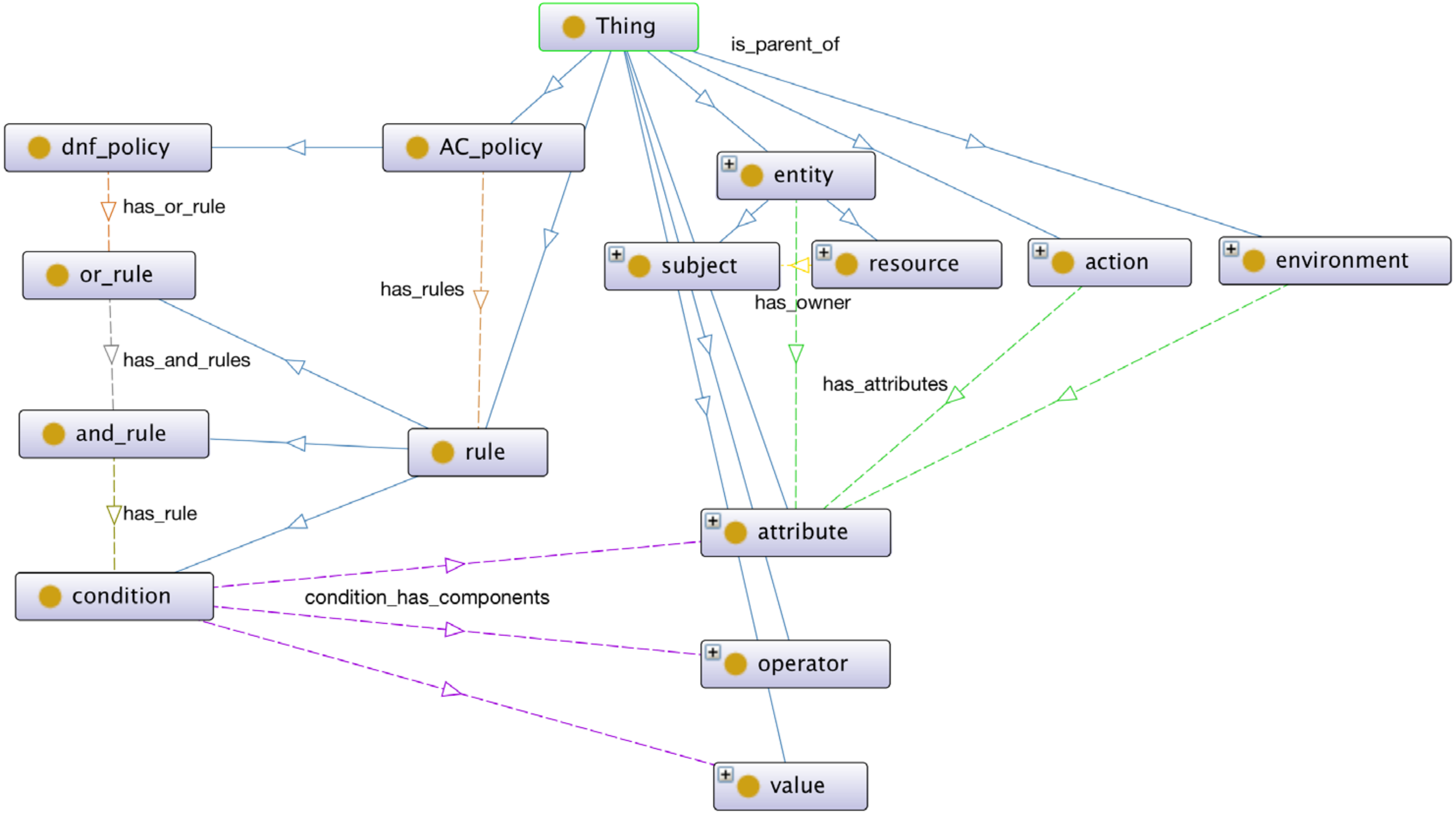
Support for Attribute Hierarchies

- Some attributes naturally have a hierarchy of values e.g. roles.
- it supports attribute hierarchies in the ***value*** and ***hierarchy*** tables that show the superior/subordinate relationships between values.
- For clouds that do not support attribute hierarchies (e.g. OpenStack) then the mapping adaptor can replace a subordinate value with it and all its superiors (so that the latter will inherit the subordinate’s properties).

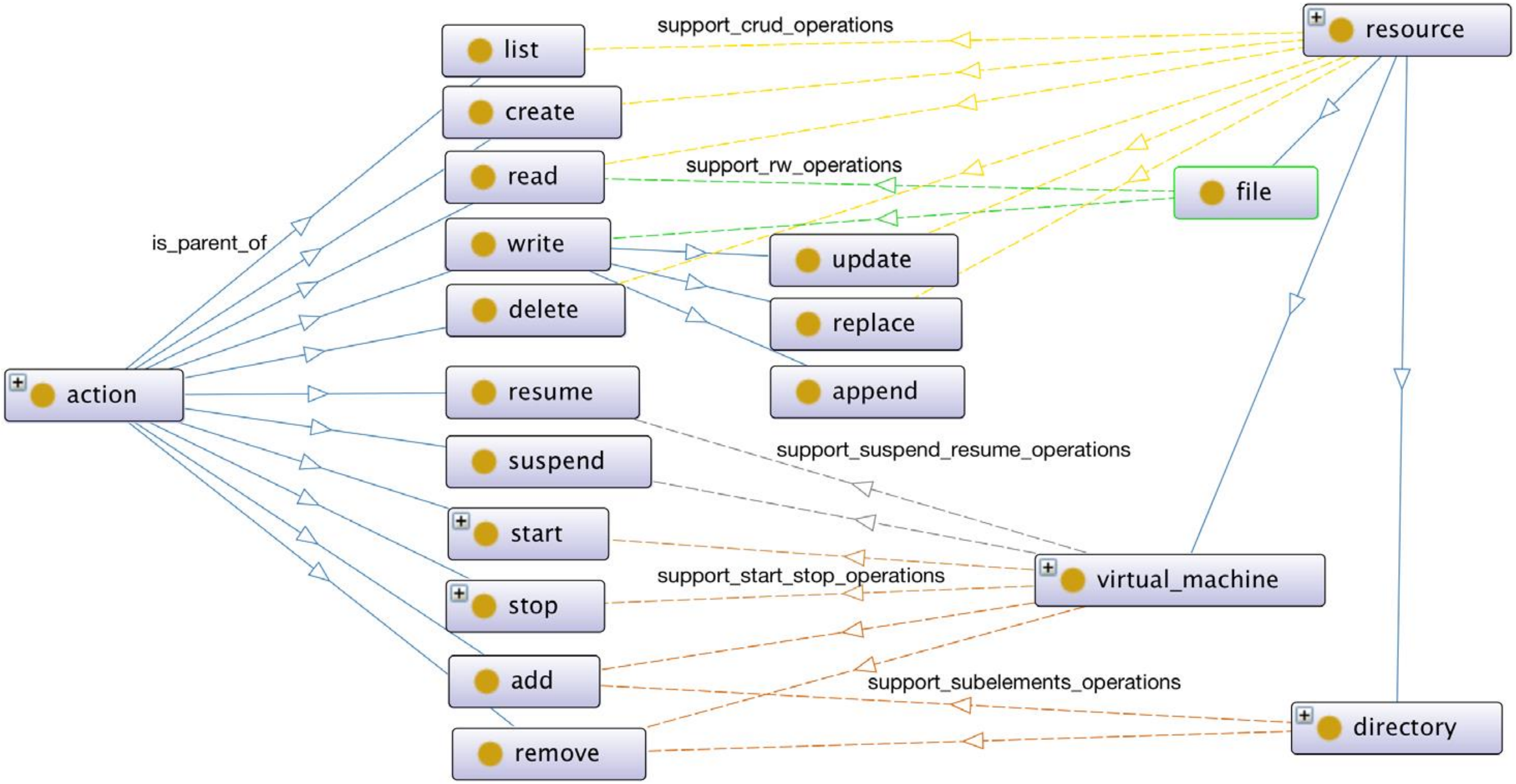
Support for Cloud Specific Rules

- Some policy rules may only apply to one type of cloud, or a cloud in one admin domain
- We would still like to represent these in the abstract policy
- In this case the rules are not converted into the abstract ontology, but the attributes and/or operators are kept "as is" and are flagged in the ***cloud_technology*** table as such

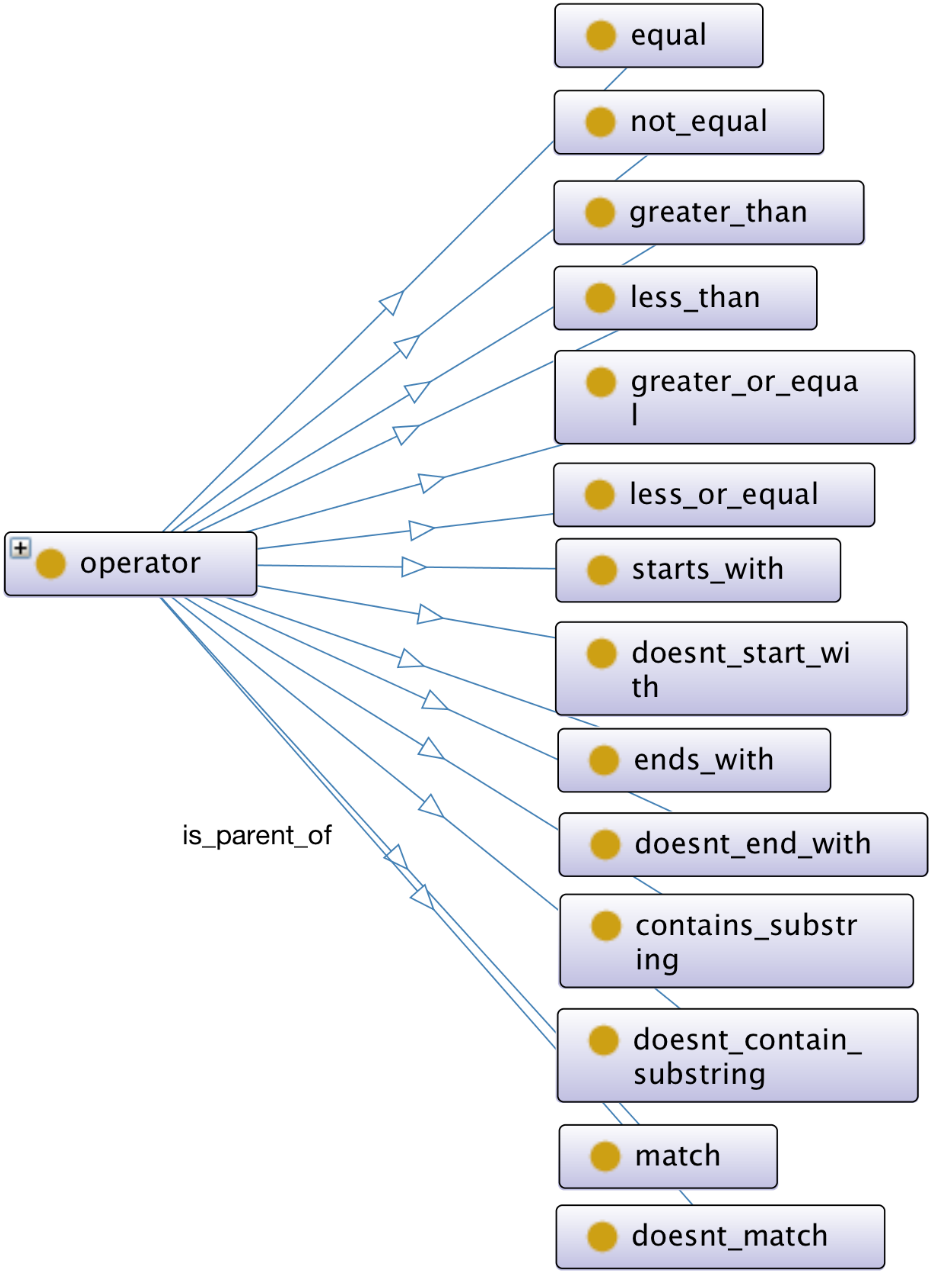
Policy Ontology



Action Ontology



Operator Ontology



API

- Policy API
- Rule API
- Search API
- Attribute API

Adaptors

- Perform syntactic mapping from cloud technology specific language to DNF and vice versa
- Perform semantic mapping from cloud technology specific terms to the ontology and vice versa, using mapping rules stored in a DB

Adaptors (Cont.)

- Two operations
  - Policy to DNF, translates a local policy into DNF
  - Policy to local, translates abstract DNF policy to a local format
- Two implementations have been built
  - Amazon Web Services policies
  - OpenStack authorization policies

OpenStack Implementation

- OpenStack authentication policy is RBAC based, and rules comprise key:value pairs, written in JSON and stored in a text file
- Rules *typically* take the form "<service>:<action>_<resource>":"<subject>"
  - E.g. "identity:update_region":"role:admin or is_admin:1"
- Adaptor syntactically maps the rules into one or more DNF 'and' rules
  - E.g. service = identity ^ action = update ^ resource = region ^ role = admin V service = identity ^ action = update ^ resource = region ^ is_admin = 1

AWS Implementation

- Amazon policies are written in JSON, and comprise two types
  - User based policies attached to subjects (e.g. users, groups, roles)
  - Resource based policies attached to resources
- Both need to be combined in the DNF
- AWS policies are much more complex than OpenStack ones
  - Grant and Deny rules, separate rules on Subjects, Actions, Resources and Environment, wildcards and variables for values, …
- Resources and roles are named using Amazon Resource Names (ARNs) which take the general form "arn:<Partition>:<Service>:<Region>:<Account>:<Resource>"
  - E.g. "arn:aws:dynamodb:us-east-1:1234567890:table/t1"

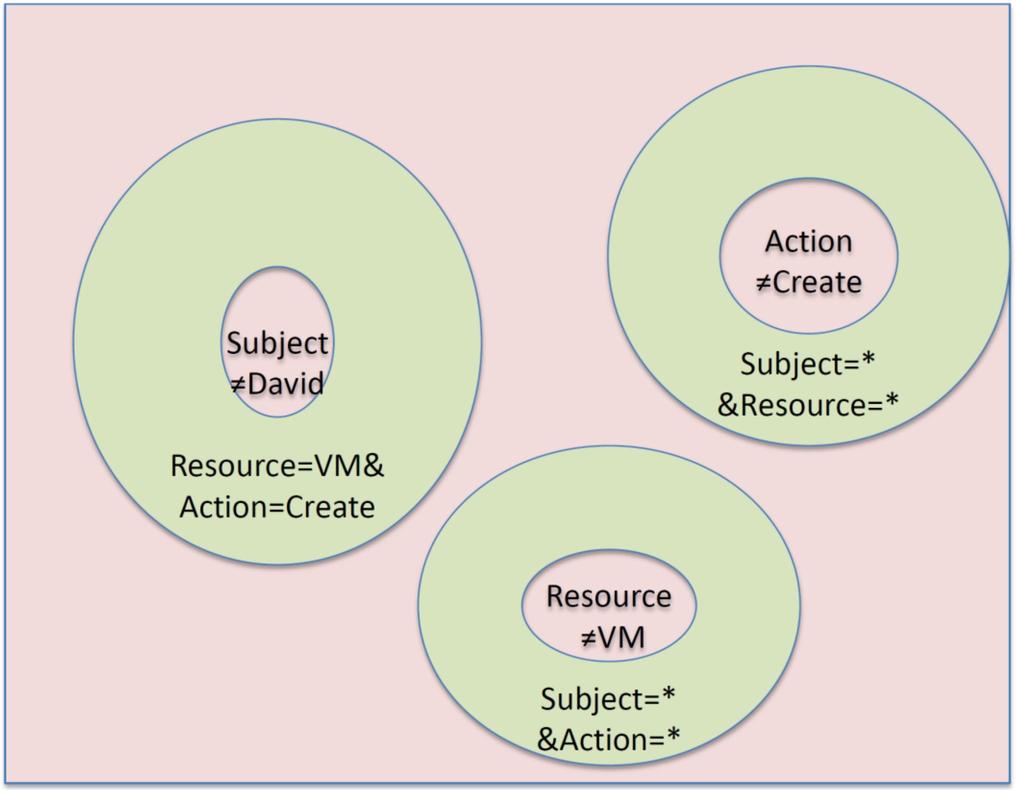
Requirements to Join FAPManS

- Provide an adaptor service that:
  - translates between the local policy and the abstract DNF and vice versa and
  - maps local policy elements to the common ontology, and vice versa
- Provide a synchronization agent that: receives notifications from FAPManS when the abstract policy is updated,
  - receives notifications from the local cloud when its local rules policy have been updated
  - uses the adaptor service to update the local cloud policy when FAPManS is updated
  - uses the adaptor service to update the local rules in FAPManS when the local cloud policy is updated (and flags an error if a federation rule has been modified)

Current Limitations

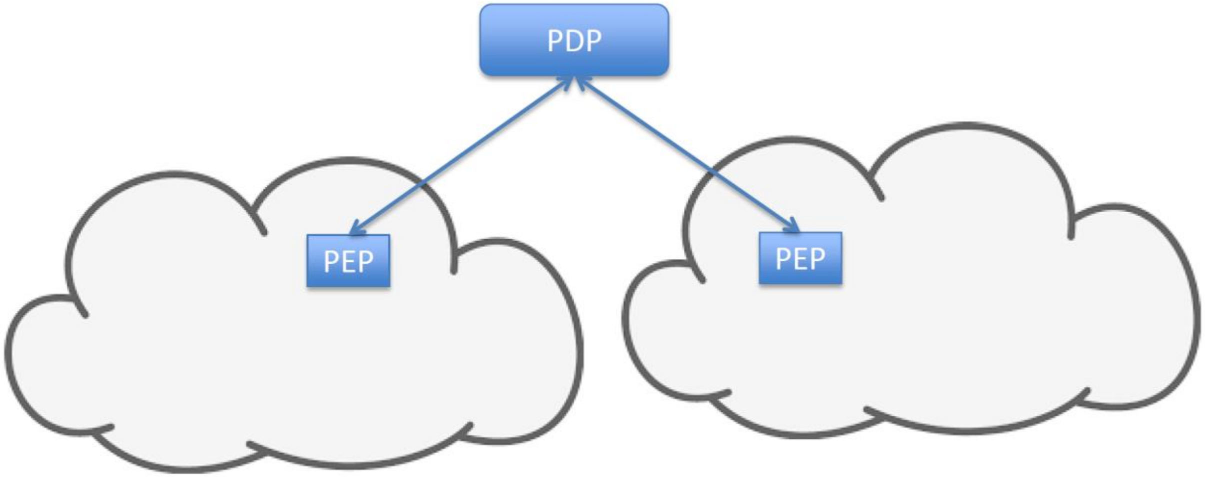
- Explicit deny rules are lost
- Mapping of non-enumerable attribute values currently not supported as its an infinite set
  - Mapping functions could be implemented to support them
- Policy Ontology/Schema is static – should be dynamically extensible
  - split the ontology definitions into two tables, named core and extensions and flag extensions as active or dormant
- Incremental merging of policies currently not supported

Venn Diagram Representation of Policies



Alternative Design

- Centralized PDP that all the federated clouds talk to for authorisation decisions



Comparison

Centralized PDP

- Central point of failure
- Bottleneck to performance
- Intrusive to normal operation of cloud service
- Homogenous policy across all clouds

FAPManS

- No central point of failure
- No performance change as cloud authorisation decision making is not altered
- Requires a lot of machinery to implement it
- Common abstract policy can only be the intersection of local cloud policies

Thank You