Explication du modèle:

```
[request_definition]
r = dom, sub, obj, act

[policy_definition]
p = dom, sub, obj, act

[role_definition]
g = _, _
g2= _, _
g3= _, _, _

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = r.dom== p.dom && (r.obj.Name== p.obj || g(r.sub, p.sub)) && r.act == p.act && (g2(r.obj.Name, p.obj) || g3(r.sub, p.sub, r.dom)) ||
r.sub == r.obj.Owner && r.dom == r.obj.Domain && r.act == 'read' ||
r.sub == r.obj.Creator && r.dom == r.obj.Domain
```

Pour faire une requête, il faut respecter la manière dont une requête est définie, l'ordre est primordial:

Requête = domaine, sujet, objet, action

De même pour définir une politique (on peut aussi définir une deuxième politique p2), et on définit une politique pour créer des rôles métiers dans des domaines afin de leurs donner des accès à des données.

Pour la définition des rôles, j'ai utilisé g pour associer un sujet (une personne ou un process) à un groupe, notamment, j'ai utilisé g2 pour définir une hiérarchie entre les ressources et les domaines, et enfin g3 pour lier des utilisateurs à des groupes dans des domaines.

D'abord, les objets dans les requêtes vont être de type « **Struct** », qui va contenir le nom de la donnée, son créateur, son propriétaire et son domaine de définition.

Pour la partie des matchers :

Nous avons définis 3 conditions, les deux dernières servent à définir des accès ABAC selon l'attribut de la personne, pour ceci, nous avons décidé d'autoriser tout accès pour les créateurs des ressources sauf qu'il peut l'accéder dans son propre domain, par contre, pour les propriétaires des ressources, on va les laisser juste le droit de lire par exemple (ceci est juste un exemple pour tester et on peut définir plusieurs attributs et même changer la manière dont on définit un sujet si on veut pour avoir plus d'extensivité).

Explication des exemples :

L'utilisateur "devops_evangelist" de l'organisation "orness" a des permissions "exec" sur les ressources "data1" dans le domaine "orness" et "domain1. Sub1".

L'utilisateur "automation_architect" de l'organisation "orness" a des permissions "exec" sur les ressources "data2" dans le domaine "orness" et "domain1. Sub2".

L'utilisateur "release_manager" de l'organisation "orness" a des permissions "exec" sur les ressources "data3" dans le domaine "orness" et "domain2. Sub1".

…

L'utilisateur "devops_evangelist" de l'organisation "ditrit" a des permissions "exec" sur les ressources "data1" dans le domaine "ditrit".

L'utilisateur "automation_architect" de l'organisation "ditrit" a des permissions "exec" sur les ressources "data2" dans le domaine "ditrit".

L'utilisateur "release_manager" de l'organisation "ditrit" a des permissions "exec" sur les ressources "data3" dans le domaine "ditrit".

L'utilisateur "software_developer" de l'organisation "orness" a des permissions "write" sur les ressources "data4" dans le domaine "orness".

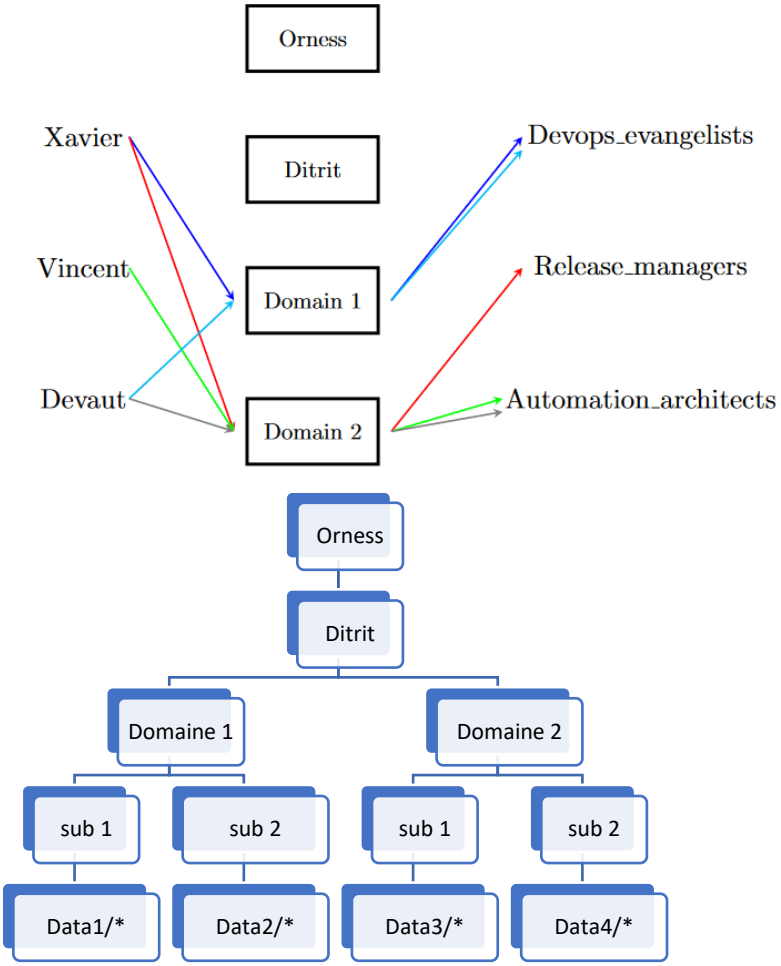
L'utilisateur "Xavier" appartient au groupe "devops_evangelists" du domaine "domain1. Sub1".

L'utilisateur "Xavier" appartient au groupe "release_managers" du domaine "domain2. Sub1".

L'utilisateur "Vincent" appartient au groupe "automation_architects" du domaine "domain1. Sub2".

L'utilisateur "Devaut" appartient au groupe "devops_evangelists" du domaine "domain1. Sub1".

L'utilisateur "Devaut" appartient au groupe "automation_architects" du domaine "domain1. Sub2".



Les tests :

Pour la partie ABAC :

Orness,Pierre,{Name:"test",Owner:"Xavier",Domain:"Orness", Creator: "Pierre"}, **write** T

Orness,Pierre,{Name:"test2",Owner:"Pierre",Domain:"Orness", Creator: "Xavier"}, **write** F

Orness,Pierre,{Name:"test2",Owner:"Pierre",Domain:"Orness", Creator: "Xavier"}, ***read*** T

Orness,Pierre,{Name:"test3",Owner:"Xavier",Domain:"Ditrit", Creator: "Pierre"}, **write** F

Ces quatres tests ont pour but de tester si ABAC fonctionne très bien avec nos matchers, pour ceci, le premier test vérifie si Pierre - le créateur de la ressource « test » que Xavier possède dans le domaine Orness - a le droit d'écrire, ce qui retourne vrai et c'est ce qu'on veut. En général, le modèle fonctionne bien, il nous permets aussi d'interdire des accès à des ressources dans d'autre domaines que celui dont la ressource est définie.

Pour la partie RBAC avec domaines :

domain1.sub2,Vincent, {Name:"data2",Owner:"Xavier",Domain:"Orness", Creator: "Pierre"}, exec

domain1.sub2,Vincent, {Name:"data3",Owner:" Xavier ",Domain:"Orness", Creator: "Pierre"}, exec

domain1.sub1, Vincent, {Name:"data2",Owner:" Xavier ",Domain:"Orness", Creator: "Pierre"}, exec

Ces trois tests nous permettent de vérifier les accès RBAC avec domaines classiques, acceptation d'accès, refus d'accès à cause de non-conformité aux politiques (objet), refus d'accès à cause de non-conformité aux politiques (domaine).

domain1.sub1,devaut, {Name:"data1",Owner:"Xavier",Domain:"Orness", Creator: "Pierre"}, exec

domain1.sub2,devaut, {Name:"data2",Owner:"Xavier",Domain:"Orness", Creator: "Pierre"}, exec

Ces deux tests nous permettent de vérifier les accès RBAC avec domaines dont un sujet a des différents roles dans des differents domaines.

Orness, super, {Name:"domain1",Owner:"Xavier",Domain:"Orness", Creator: "Pierre"}, exec

Ce dernier test est un test d'hiérarchie entre les domaines (et les ressources), comme nous avons permis à super d'accéder à Ditrit dans le domaine Orness, comme domain1 est un sous domaine de Ditrit, super doit pouvoir accéder, le résultat de ce test est Vrai.