Modelos RBAC [1]

Notas para a UC de “Segurança Informática”

Inverno de 11/12

José Simão (jsimao *em* cc.isel.ipl.pt)

Instituto Superior de Engenharia de Lisboa

Agenda

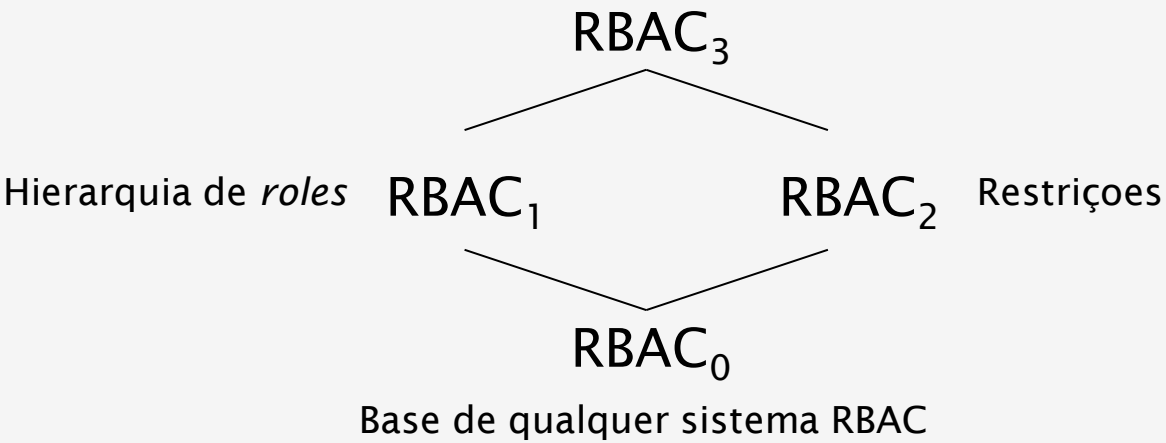
- Controlo de acessos baseado em *roles*
  - Família de modelos
    - Base: $RBAC_0$
    - Hierarquia de *roles*: $RBAC_1$
    - Restrições: $RBAC_2$
    - Unificação: $RBAC_3$
  - Administração

Modelos RBAC – Motivação

- Para efeitos de controlo de acessos é mais relevante saber as responsabilidades do utilizador do que quem ele é
- Nas organizações as permissões estão tipicamente associadas a *roles* e não a pessoas
  - Funcionário da tesouraria; Director de departamento
- As permissões associadas a *roles* mudam com menos frequência do que a associações entre utilizadores e *roles*
- As organizações valorizam princípios como o da separação de poderes
  - e.g. Quem pede material para projectos não deve ser a mesma pessoa que autoriza o pagamento

Família de modelos RBAC

- São quatro os modelos da família RBAC



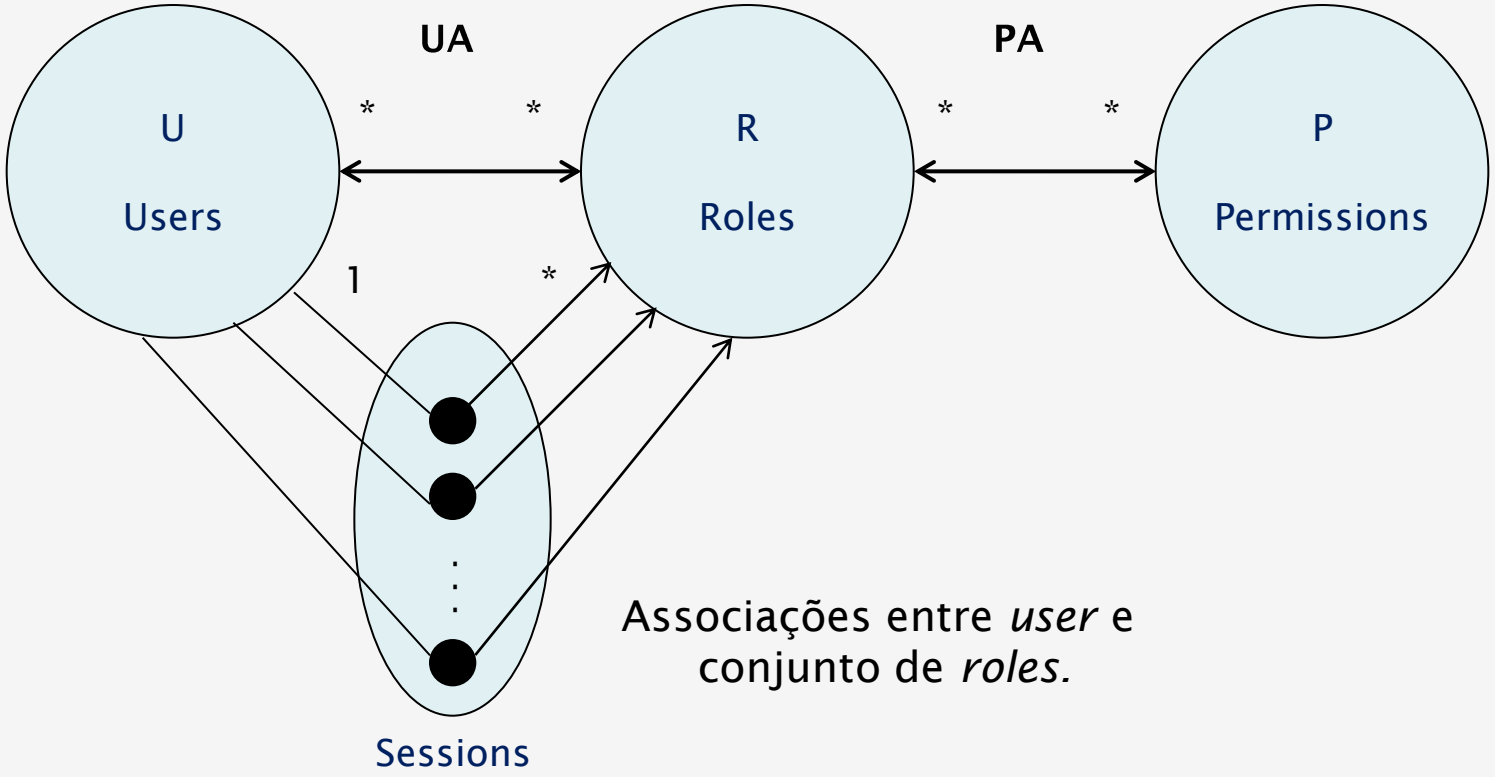
- Servem de referência para caracterizar os sistemas que usam este tipo de controlo de acessos

$RBAC_0$

Conjunto de utilizadores

Actividades desempenhadas dentro da organização

Autorização para aceder a um ou mais objectos



Associações entre *user* e conjunto de *roles.*

$RBAC_0$

- As relações *user assigment* (UA) e *permission assigment* (PA) são a base do modelo
  - Ambas são relações de muitos para muitos
- As políticas são expressas pela concretização destas relações
- As permissões são sempre positivas
- As permissões de um utilizador são a reunião das permissões dos roles activos na sessão
  - Relação de um para um entre utilizadores e sessão
  - A cada sessão pode estar associado um ou mais *roles*
  - Cada utilizador activa apenas o conjunto de roles que lhe interessa

$RBAC_0$ – Definições

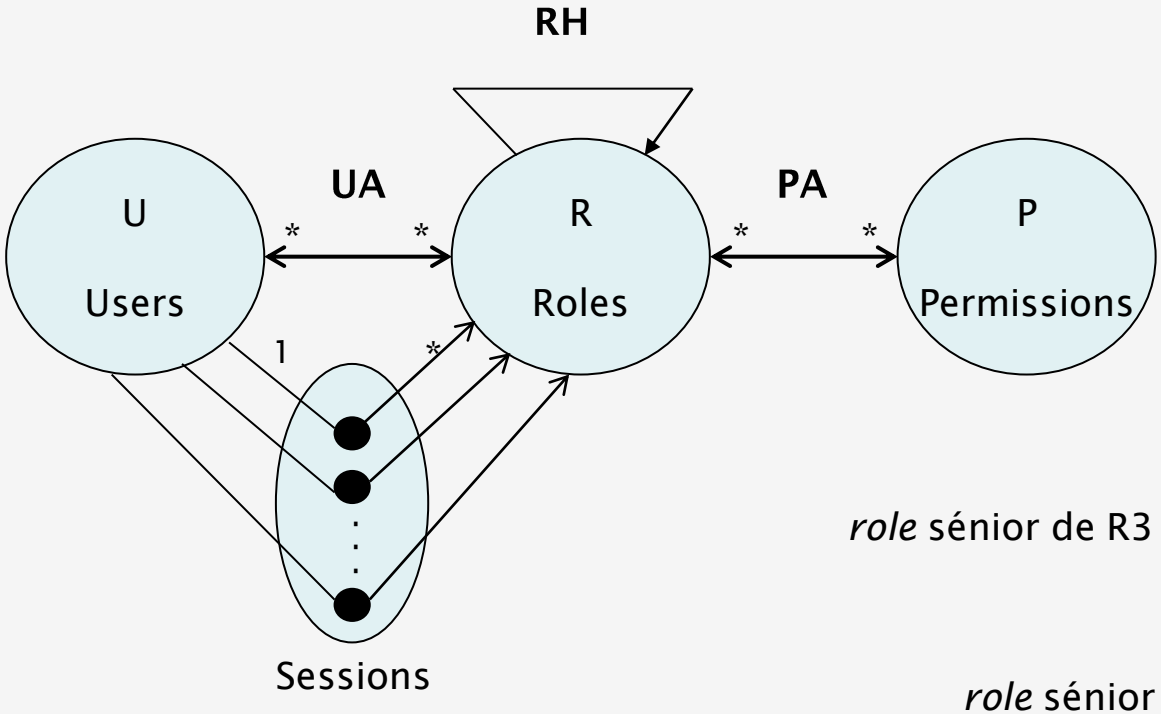
- $\boldsymbol{U}$, $\boldsymbol{R}$, $\boldsymbol{P}$ e $\boldsymbol{S}$ (*users*, *roles*, *permissions* e *sessions*)
- $\boldsymbol{UA \subseteq U \times R}$, relação de muitos para muitos entre *users* e *roles*
  - $U = \{u_1, u_2\}$ $R = \{r_1, r_2\}$
  - $UxR = \{ (u_1, r_1), (u_1, r_2), (u_2, r_1), (u_2, r_2) \}$
- $\boldsymbol{PA \subseteq P \times R}$, relação de muitos para muitos entre *permissions* e *roles*
- $\boldsymbol{user : S \rightarrow U}$, função que associa cada sessão $\boldsymbol{s_i}$ a um utilizador $\boldsymbol{user(s_i)}$ (constante ao longo da sessão)
- $\boldsymbol{roles : S \rightarrow 2^R}$, função que associa cada sessão $\boldsymbol{s_i}$ a um conjunto de *roles*

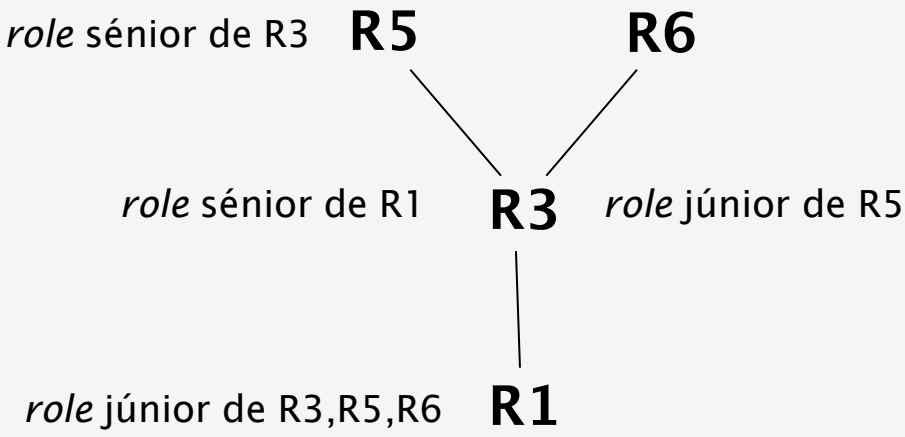
  $\boldsymbol{roles(s_i) \subseteq \{ r \mid (user(s_i), r) \in UA \}}$ (pode mudar ao longo da sessão)
  - 2^R = *power set de R, conjunto dos subconjuntos de R*

    $\{ \{\}, \{r_1\}, \{r_2\}, \{r_1, r_2\} \}$
- Cada sessão $\boldsymbol{s_i}$ tem as permissões $\boldsymbol{\bigcup_{r \in roles(s_i)} \{ p \mid (p, r) \in PA \}}$

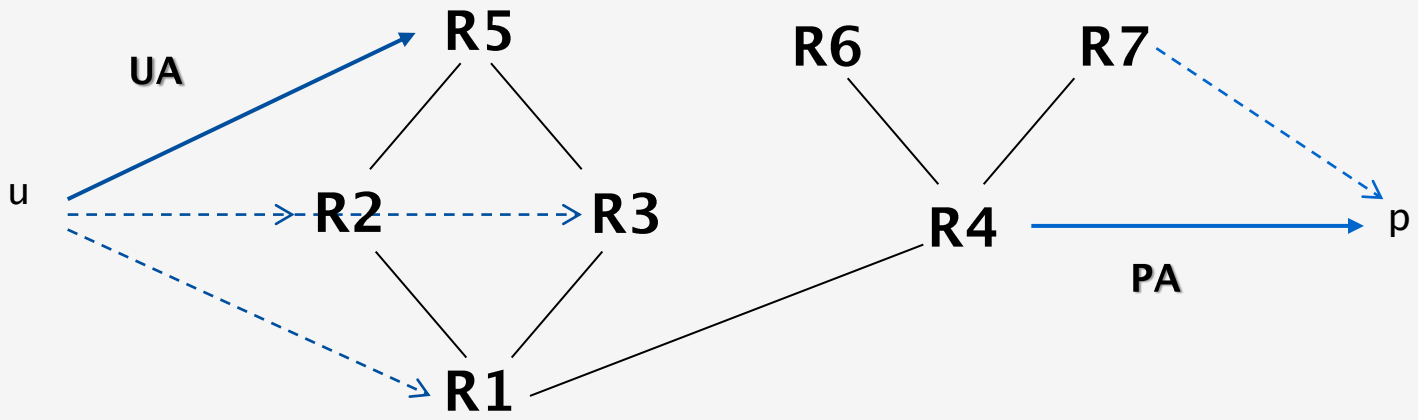
$RBAC_1$

- Introduz o conceito de hierarquia de *roles*





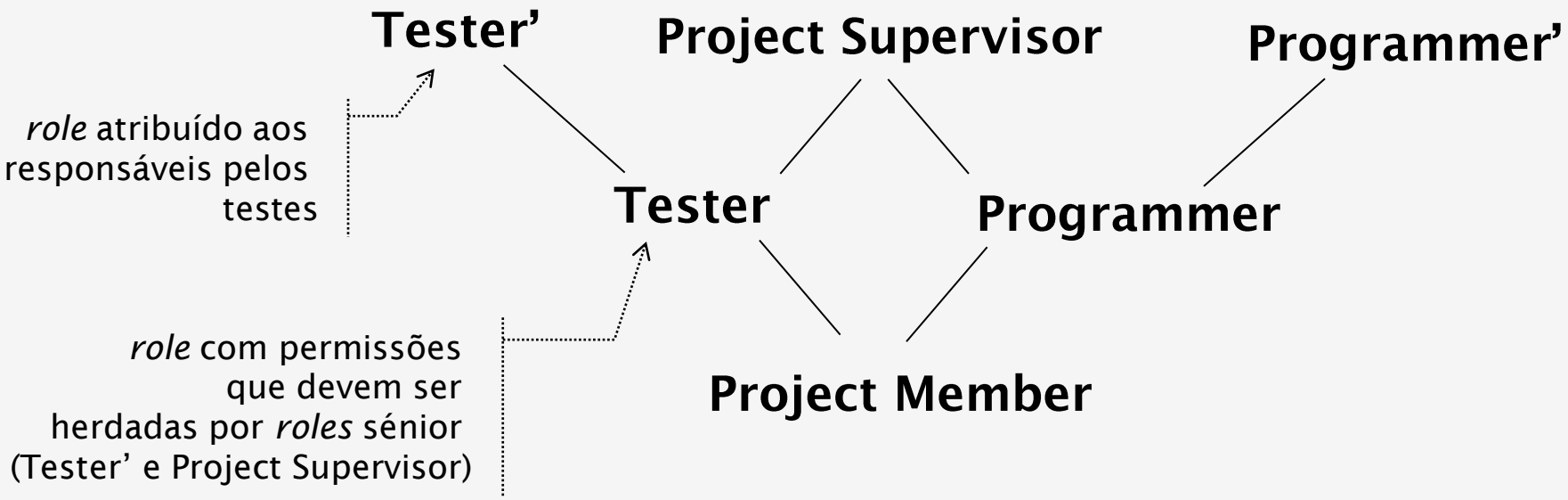
$RBAC_1$



- O utilizador escolhe qual o *role* que quer activar, herdando os *roles* júnior desse
- As permissões são as directamente associadas ao *role* do utilizador mais as dos roles júnior

$RBAC_1$

- Pode ser útil limitar os *roles* herdados (*private roles*)
  - e.g. Resultados intermédios de trabalho em progresso podem não fazer sentido serem analisados pelo responsável de projecto



$RBAC_1$ – Definições

- U, R, P, S, UA, PA
- $RH \subseteq R \times R$, relação de dominância entre roles, representada por $\geq$
- $roles : S \rightarrow 2^R$, função modificada de $RBAC_0$ para ter como requisito
  $roles(s_i) \subseteq \{ r \mid (\exists r' \geq r) [(user(s_i), r') \in UA] \}$ (pode mudar ao longo da sessão)
- Cada sessão s_i tem as permissões
  $\bigcup_{r \in roles(s_i)} \{ p \mid (\exists r'' \leq r) [(p, r'') \in PA] \}$

$RBAC_2$

- Restrições são um mecanismo para impôr regras da organização
  - Podem ser aplicadas às relações UA e PA, e às funções *user* e *roles*
- Têm a forma de predicados, retornando “aceite” ou “não aceite”
- Exemplos de restrições
  - Separação de deveres
    - Pode ser garantido de forma estática (relação UA) ou dinâmica (função *roles*)
  - Cardinalidade
  - Pré-requisitos

$RBAC_3$

- Combina $RBAC_1$ e $RBAC_2$ suportando hierarquia de *roles* e restrições

- Num cenário onde a administração é delegada em terceiros pode ser necessário impôr restrições
  - E.g. Dois ou mais *roles* sénior não podem ter em comum determinados *roles* júnior

- Nos *roles* privados, os que servem de base (*Tester* e *Programmer*, na Figura da pág.13) podem ter cardinalidade máxima de zero elementos

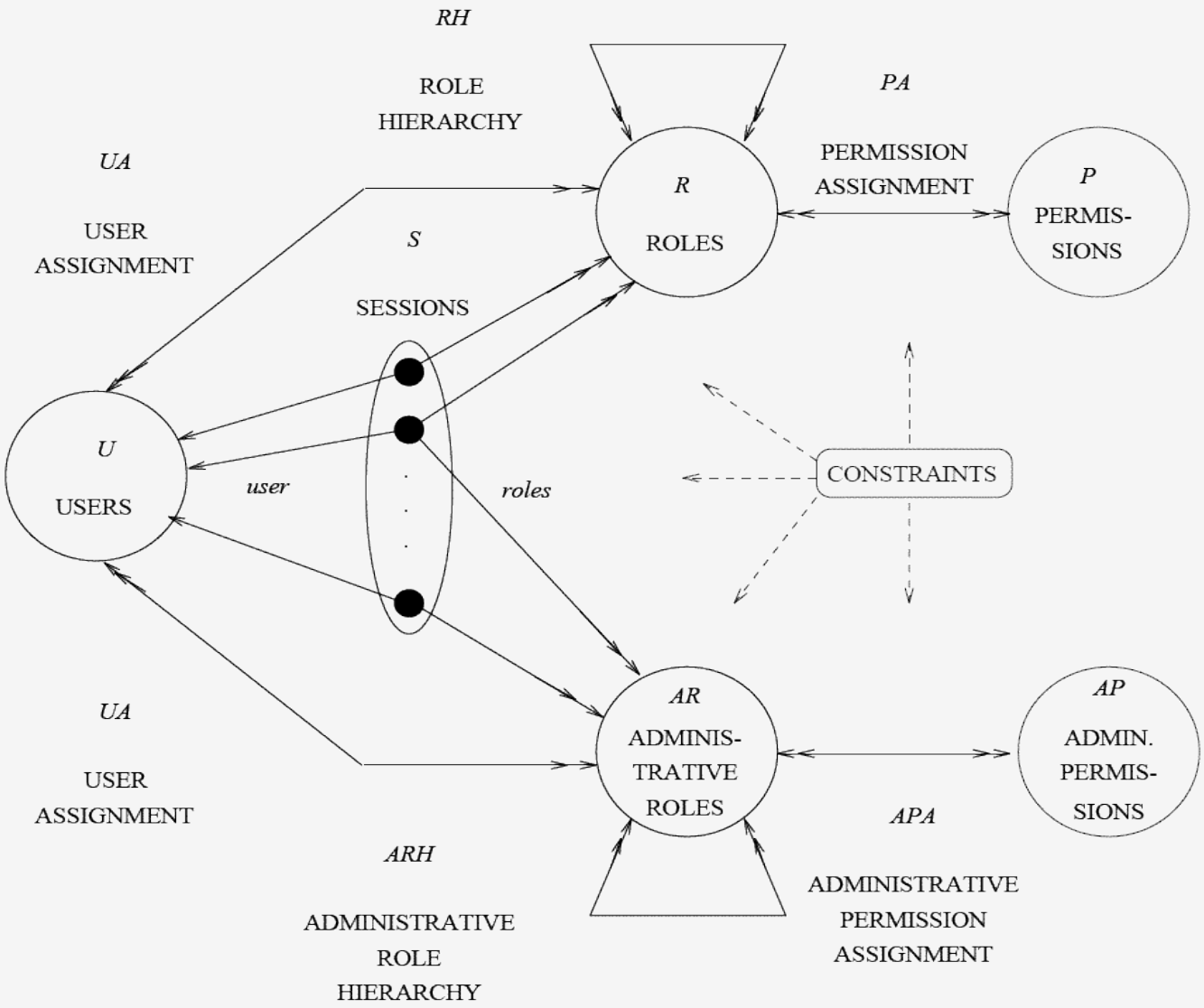
Modelo de administração – ARBAC

- O próprio modelo RBAC pode ser utilizado para promover a descentralização do processo de gestão, estabelendo *roles* e permissões de administração

- As permissões de administração traduzem-se na possibilidade de modificar as associações UA e PA para um domínio
  - Aos utilizadores com roles de administração são atribuídas permissões de administração

- As restrições são aplicáveis a ambos os modelos (RBAC e ARBAC)
  - Se a relação UA tiver exclusão mútua sobre *roles* normais e *roles* de administração for por exclusão mútua, garante-se que é possível gerir os componentes RBAC, mas não é possível usar as permissões

Figura geral dos modelos RBAC



Conclusões

- RBAC ≠ grupos
  - Um *role* representa um conjunto de permissões e não um conjunto de utilizadores
  - Os utilizadores escolhem qual o role que pretendem desempenhar
- O modelo suporta cenários simples e complexos
- Dada a natureza neutral quanto ao tipo de políticas impostas pelo modelo, este pode ser usado para a sua própria gestão

Caso prático: Suporte à autorização por *roles* em .NET

- A plataforma .NET fornece serviços de autorização baseada em *roles* tendo por base uma arquitectura baseada em *providers*.
- Cada role provider tem de derivar da classe abstracta RoleProvider, que define uma interface genérica para a gestão de roles.
- A plataforma .NET oferece três *role providers*:
  - SQL
  - Windows
  - AzMan
- É possível utilizar os serviços da plataforma .NET com *role providers* personalizados se estes derivarem da classe RoleProvider.
- O RoleProvider por omissão está disponível através da classe estática Roles.

Classe RoleProvider

- A classe abstracta RoleProvider define a seguinte interface de gestão de roles:
  - AddUsersToRoles
  - CreateRole
  - DeleteRole
  - FindUsersInRole
  - GetAllRoles
  - GetRolesForUser
  - GetUsersInRole
  - IsUserInRole
  - RemoveUsersFromRoles
  - RoleExists

Exemplo da configuração de um role provider

- A configuração é feita no ficheiro de configuração da aplicação.

```
<system.web>
    <connectionStrings>
        <add name="SqlServices" connectionString="..." />
    </connectionStrings>

    <roleManager defaultProvider="SqlProvider" >
        <providers>
            <add
                name="SqlProvider"
                type="System.Web.Security.SqlRoleProvider"
                connectionStringName="SqlServices"
                applicationName="SampleApplication" />
        </providers>
    </roleManager>
</system.web>
```

Referências

[1] R. S. Sandhu, et al. “Role-Based Access Control Models”, IEEE Computer 29(2): 38-47, IEEE Press, 1996.