2
Fundamentos

2.1.
Controle de Acesso

O mecanismo de controle de acesso inclui dois conceitos fundamentais e distintos, porém dependentes: a autenticação e a autorização [Ferraiolo et al., 2007].

A autenticação é o processo de verificar se a identidade do usuário é legitima, isto é, se o usuário é realmente quem diz ser. Ela é baseada em três fatores: alguma coisa que o usuário sabe, como uma senha, a combinação de um cadeado; alguma coisa que o usuário tem, como um cartão ou uma chave; ou em alguma característica física, como a impressão digital, a voz ou a retina.

Já a autorização consiste em determinar o que um sujeito pode fazer no sistema. A autorização é usada para definir as políticas de acesso através de regras de controle de acesso. Um sistema de computador usa uma regra de controle de acesso para decidir se é permitido ou negado o acesso a um recurso baseado na identidade obtida durante a autenticação. Desta forma, é importante observar que a autorização depende de uma autenticação feita de forma apropriada.

Diversas terminologias para a descrição dos modelos de controle de acesso tem sido desenvolvidas durante as últimas três décadas. Qualquer modelo de controle de acesso pode ser descrito formalmente usando descrições de usuários, sujeitos, objetos, operações e permissões, e os relacionamentos entre estas entidades [Ferraiolo et al., 2007]. Uma breve descrição destes termos será apresentada a seguir.

- **Usuários**: referem-se tipicamente as pessoas que interagem com o sistema de computador, mas também podem representar outros agentes como algum dispositivo ou uma máquina.
- **Sujeitos**: são processos ou tarefas que agem em nome do usuário e executam uma operação ou uma série de operações sobre um ou mais objetos. Dois ou mais sujeitos podem corresponder a um mesmo usuário, mesmo que este tenha apenas um *login* e esteja

na mesma sessão. Uma sessão é uma instância de diálogo do usuário com o sistema. Todos os sujeitos têm um identificador único.

- **Objetos**: são entidades do sistema sobre as quais as operações são executadas. No contexto de um Sistema Operacional, um objeto pode representar um arquivo. No contexto de um sistema gerenciador de banco de dados, um objeto pode ser uma tabela ou uma *view*. Outros exemplos de objetos são: *buffers*, diretórios de arquivos, páginas da *web*, programas, impressora, etc. A escolha de quais entidades que irão pertencer ao conjunto de objetos é determinada pelos requerimentos de proteção e objetivos de segurança do sistema.
- **Operações**: são processos ativos invocados pelo sujeito. Exemplos de operações no contexto de um sistema bancário podem ser: fazer depósito, sacar, verificar o saldo, entre outros.
- **Permissões (ou privilégios)**: são direitos dados a um individuo, ou ao sujeito agindo em nome do usuário, permitindo ao detentor do direito executar alguma ação no sistema. O termo permissão refere-se a uma relação entre o objeto, o sujeito, e a operação.

Na próxima seção será descrito brevemente o modelo *Role-based Access Control* (RBAC).

2.2. Role-based Access Control model

Segundo Ferraiolo et al. (2007), os papéis dos usuários dentro de uma organização são relativamente estáveis, enquanto que os usuários e as permissões são ambos numerosos e podem mudar com frequência. O modelo *Role-based Access Control* (RBAC) fornece uma maneira segura e eficiente para gerenciar o acesso às informações de uma organização que prescinde da enumeração explicita de todos os usuários e permissões, reduzindo a complexidade e os custos administrativos, e minimizando os erros. O principal objetivo do RBAC é facilitar a análise e o gerenciamento do controle de acesso. Para os autores, o controle de acesso baseado em papel é perfeitamente adequado para uma grande variedade de aplicações e ambientes, em particular as aplicações comerciais, e suportam uma grande diversidade de políticas de

controle de acesso necessárias para atender os domínios de aplicações do mundo real.

Ferraiolo e Kuhn (1992, 1995) definiram três regras básicas para o modelo RBAC. A primeira define que um sujeito só pode executar uma transação se ele tiver sido associado a um papel. A segunda regra diz que um sujeito só pode ativar aqueles papéis que são autorizados para ele. Por fim, a terceira regra determina que um sujeito pode executar somente aquelas transações que são autorizadas para o papel dele. Outra importante característica definida pelos autores é que os papéis podem ser organizados em uma hierarquia de permissões, podendo assim herdar as permissões de outros papéis. Isto torna a especificação das permissões em uma aplicação mais sucinta, devido à fatoração de permissões comuns a diversos papeis em um papel hierarquicamente superior a todos eles.

As características do RBAC variam de simples para complexas. Para distinguir as características incorporadas nos vários modelos RBAC propostos na literatura, uma taxonomia foi desenvolvida [Sandhu et al., 1996] e consiste em quatro modelos: RBAC nuclear, RBAC hierárquico, RBAC com restrições estáticas e RBAC com restrições dinâmicas. Essa divisão pode ser chamada também de RBAC96.

O modelo RBAC nuclear cobre o conjunto de características básicas presentes em todo sistema que implementa o RBAC. São estas características que distinguem um sistema RBAC de outras formas de gerenciamento de controle de acesso. O modelo RBAC hierárquico acrescenta o conceito de hierarquia de papéis, enquanto que o modelo RBAC com restrições estáticas adiciona restrições impostas nas alocações dos papéis e o modelo RBAC com restrições dinâmicas impõem restrições na ativação do papel de um usuário. O modelo RBAC com restrições estáticas e dinâmicas não será abordado neste trabalho.

2.2.1. Modelo RBAC Nuclear

O modelo RBAC nuclear identifica cinco elementos importantes: (1) usuário, (2) papel do usuário e (3) permissões que são compostas por (4) operações aplicadas a um ou mais (5) objetos. A principal característica deste modelo é que todo acesso a algum objeto do sistema é feito através do papel ou função do usuário. O papel é essencialmente uma coleção de permissões e

todos os usuários recebem permissões somente através dos papéis aos quais os usuários são alocados. Desta forma, quando atribuímos um papel a um conjunto de usuários, isto é, tornamos estes usuários membros deste papel, isto irá permitir a estes usuários executar as permissões de acesso que estão associadas ao papel. A Figura 1 mostra essa relação. Note que a seta nos dois lados do relacionamento indica que o relacionamento é de muitos-para-muitos.

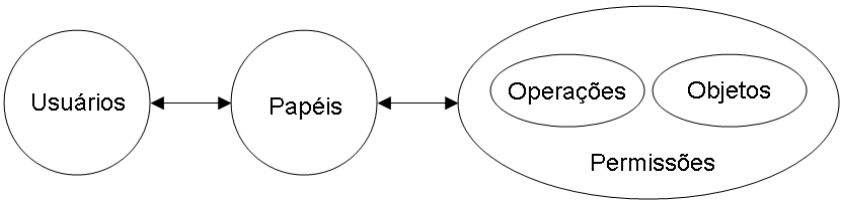

Figura 1 – Relacionamento entre os usuários, os papéis e as permissões

Através do modelo RBAC nuclear, os usuários são associados aos papéis baseando-se em suas competências, autoridades, responsabilidades, etc. Caso um usuário mude de posição dentro da organização, somente a associação com o papel dentro do modelo RBAC é mudada. Consequentemente, não é necessário remover as permissões existentes e associar um conjunto completamente novo de permissões. Sendo assim, o gerenciamento das permissões tem a grande vantagem de ser simplificado na medida que os administradores de segurança do sistema podem atualizar os papéis dos usuários sem ter que atualizar as permissões para cada usuário.

Na modelagem do sistema de controle de acesso, os administradores de sistema podem tratar os papéis no mesmo nível de abstração em relação aos processos de negócio da empresa. Em um sistema bancário, exemplos de papéis podem ser: gerente, subgerente, caixa, contador, analista de crédito, supervisor, etc. Já as permissões referem-se à combinação de uma operação com um objeto, e elas refletem as decisões das políticas de acesso da empresa. Em um sistema hospitalar, o papel de um médico pode ter a permissão de prescrever medicamentos, fazer diagnósticos, solicitar testes de laboratório, etc. Em um sistema bancário, o papel de um caixa pode executar a operação de fazer depósito, pagamentos, etc.

Ferraiolo et al. (2007) dividiram as propriedades e os mapeamentos do modelo RBAC nuclear em dois componentes separados e dependentes: o componente estático e o dinâmico. O componente estático do modelo RBAC é definido por todos os termos e relacionamentos que não envolvem a noção do sujeito, como mostrado na Figura 1. Vale lembrar que o sujeito é uma entidade ativa que age em nome do usuário e realiza todas as solicitações deste. Cada

sujeito possui um identificador único e um usuário pode estar associado a um ou mais sujeitos ao mesmo tempo.

O componente dinâmico do modelo RBAC inclui a ativação do papel e o acesso ao sujeito. Para o usuário poder usar as permissões associadas ao seu papel, este papel precisa antes ser ativado. A ativação do papel é feita pelo sujeito. O sujeito só pode ativar aqueles papéis autorizados a ele, isto é, os papéis ativos devem ser um subconjunto dos papéis que são associados ao usuário deste sujeito. Cada sujeito pode ter diferentes combinações de papéis ativos. Os componentes estático e dinâmico são necessários para um usuário acessar um objeto.

A Figura 2 ilustra estes dois componentes. As linhas tracejadas referem-se aos mapeamentos dinâmicos e as linhas contínuas referem-se aos relacionamentos estáticos. A Figura 2 também mostra que o usuário u_1 pode executar a operação op_2 no objeto o_2 porque $p_2 \in$ assigned_permissions(r_2) $\wedge$ $u_1 \in$ assigned_users(r_2) $\wedge$ $u_1 \in$ sujeito_usuário(s_2) $\wedge$ $r_2 \in$ sujeito_papéis(s_2).

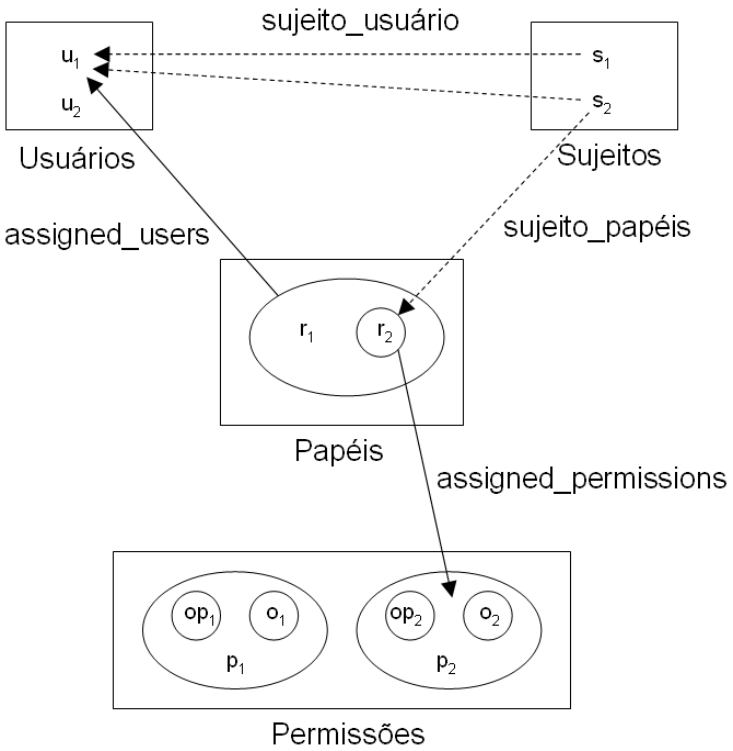

Figura 2 – Componente estático e dinâmico do modelo RBAC nuclear

O modelo RBAC nuclear de Ferraiolo pode ser formalmente definido da seguinte forma:

- Seja *USUÁRIOS* o conjunto dos usuários, *PAPÉIS* o conjunto dos papéis dos usuários, *SUJEITOS* o conjunto dos sujeitos, *OPS* o conjunto das operações e *OBS* o conjunto dos objetos.
- *UA* $\subseteq$ *USUARIOS* X PAPÉIS, o relacionamento de muitos-para-muitos entre os usuários e os papéis.
- *PRMS* = $2^{(OPS \mathrm{X} OBS)}$, o conjunto das permissões.
- *PA* $\subseteq$ *PRMS* X *PAPÉIS*, o relacionamento de muitos-para-muitos entre as permissões e os papéis.
- *assigned_users* : *PAPÉIS* $\rightarrow$ $2^{USUÁRIOS}$, isto é, *assigned_users(r)* = {*u* $\in$ *USUÁRIOS* | (*u, r*) $\in$ *UA*}, representa o mapeamento do papel *r* com o conjunto de usuários.
- *assigned_permissions* : *PAPÉIS* $\rightarrow$ 2^{PRMS}, isto é, *assigned_permissions(r)* = {*p* $\in$ *PRMS* | (*p, r*) $\in$ *PA*}, representa o mapeamento do papel *r* com o conjunto de permissões.
- *sujeito_usuário* : *SUJEITOS* $\rightarrow$ *USUÁRIOS*, isto é, *sujeito_usuário(s) = u* $\in$ *USUÁRIO*, representa o mapeamento do sujeito *s* com o usuário *u* associado a ele.
- *sujeito_papéis* : *SUJEITOS* $\rightarrow$ $2^{PAPÉIS}$, isto é, *sujeito_papéis*(s_i) $\subseteq$ {*r* $\in$ *PAPÉIS* | (*sujeito_usuário*(s_i), *r*) $\in$ *UA*}, representa o mapeamento do sujeito *s* com o conjunto de papéis.

A partir destas definições, duas propriedades do modelo RBAC são definidas:

Propriedade 1 – Autorização da ativação do papel. Esta propriedade determina que o sujeito só pode ativar um papel quando este papel for autorizado pelo usuário que está associado a esse sujeito.

- Dado que *s* $\in$ *SUJEITOS*, *u* $\in$ USUÁRIOS e *r* $\in$ PAPÉIS, temos que: $(\forall s,u,r)$ (*r* $\in$ *sujeito_papéis(s)* $\wedge$ *u* $\in$ *sujeito_usuário(s)* $\Rightarrow$ *u* $\in$ *assigned_users(r)*)

Propriedade 2 – Autorização de acesso a um objeto. Esta propriedade determina que um sujeito *s* só pode executar uma operação *op* em um objeto *o* somente se existir (i) um papel *r* no qual foi ativado pelo sujeito *s,* e (ii) uma permissão *p* associada ao papel *r* autorizando a execução da operação *op* sobre o objeto *o*.

- *acesso* : *SUJEITOS* X *OPS* X *OBS* $\rightarrow$ *BOOLEAN*

- *acesso(s, op, o)* = 1, se o sujeito *s* pode acessar o objeto *o* usando a operação *op,* e 0 caso contrário.
- Dado que $r \in$ *PAPÉIS* e $p \in$ *PRMS*, temos que: *acesso(s, op, o)* $\Rightarrow$ $(\exists r, p)$ ($r \in$ *sujeito_papéis(s)* $\wedge$ $p \in$ *assigned_permissions(r)* $\wedge$ (*op, o*) $\in p$).

2.2.2. Modelo RBAC Hierárquico

A hierarquia de papéis do modelo RBAC é uma forma natural de estruturar os papéis a fim de refletir os níveis de autoridade dentro de uma organização [Sandhu et al., 2000]. Por convenção, os papéis mais poderosos (papéis *senior*), isto é, aqueles que agregam maior quantidade de direitos de acesso, são colocados no topo da hierarquia de papéis, enquanto que os menos poderosos (papéis *junior*) ficam na base da hierarquia.

A herança na hierarquia de papéis do modelo RBAC tem interpretação inversa à herança na hierarquia de classes em Orientação a Objeto. Na hierarquia de papéis, um indivíduo consegue mais privilégios quando ele move para cima na hierarquia, enquanto que na hierarquia de classes, uma classe consegue mais atributos quando ela é movida para baixo na hierarquia.

2.3. ACL (Access Control List)

ACL (*Access Control List*) é um mecanismo de controle de acesso e não um modelo, como o RBAC. O ACL é o método mais comum para se implementar o controle de acesso em um sistema na forma de uma Matriz de Controle de Acesso [Ferraiolo et al., 2007, p. 43].

A Matriz de Controle de Acesso foi o primeiro trabalho desenvolvido relacionado ao controle de acesso [Lampson, 1969]. Lampson introduziu formalmente o conceito de sujeito e objeto, e uma matriz de controle de acesso que atuava como mediador de acesso entre o sujeito e o objeto. A Matriz de Controle de acesso é uma representação conceitual simples, em que a entrada (i, j), na matriz, especifica os direitos de acesso que o sujeito i tem sobre o objeto j. Direitos de acesso tipicamente incluem direito à leitura, escrita, ou posse.

O ACL implementa a Matriz de Controle de Acesso da seguinte forma: cada objeto da matriz está associado a uma lista de usuários com seus respectivos direitos de acesso sobre o objeto. A Tabela 1 mostra um exemplo de ACL.

Objeto		
Arquivo_1	Chris: Ler, escrever	Frank: Ler
Arquivo_2	John: Executar	Barbara: Ler
Arquivo_3	Chris: Escrever	Barbara: Ler
Processo_1	John: Suspender	

Tabela 1 – Exemplo de ACL

A principal vantagem dos ACLs é sua facilidade de analisar, dado um objeto, os usuários que tem acesso a este objeto e quais operações são permitidas de serem executadas. É fácil também cancelar o acesso a determinado objeto somente pela remoção de uma entrada da lista. Outra vantagem é a possibilidade de usar grupos de usuários com acessos em comum a um objeto, ao invés de colocar cada membro individual do grupo associado ao objeto.

A desvantagem do uso do ACLs é a análise de perguntas voltadas ao sujeito como “Quais objetos o usuário X tem acesso?” ou “Cancele as permissões de acesso do usuário Y”. Para responder a estas perguntas, seria preciso fazer uma varredura de todos os objetos protegidos pelo mecanismo de controle de acesso, e analisar as ACL para cada objeto. Portanto, as ACLs são ideais para políticas de controle de acesso voltadas aos objetos.

2.4. Linguagem de Política

De acordo com Latham (1985), uma política de acesso é um conjunto de regras. Essas regras são avaliadas a fim de determinar se um usuário tem direito de acesso a um dado objeto.

Segundo Bonatti et al. (2009), a política de acesso especifica quem tem permissão de executar que ação em qual objeto dependendo de (i) propriedades do usuário que fez a solicitação, (ii) propriedades do objeto, (iii) parâmetros da ação, e (iv) fatores ambientais (como, por exemplo, o tempo). Ainda segundo os autores, política de acesso é um método bem conhecido para proteger a segurança e a privacidade dos usuários. A segurança e a privacidade são importantes para aumentar o nível de confiança dos recursos *web*.

Nos últimos anos um grande número de linguagens de políticas foram propostas, com diferentes características e expressividades, para atender uma grande variedade de cenários das aplicações. O artigo de Bonatti et al. (2009) faz um interessante estudo comparativo envolvendo doze linguagens de políticas, tais como: Rei, KAoS, XACML, Cassandra, *RT*. As comparações foram realizadas de acordo com dez critérios, entre eles: quanto à presença de semântica bem definida; quanto ao grau de expressividade das condições presentes na solicitação do usuário; se a linguagem é baseada em algum formalismo; se possui meios para delegar os direitos de acesso; se há suporte a negociação; se a linguagem de política provê algum mecanismo para extensão, entre outros.

Bonatti et al. (2009) dividiram as políticas em quatro categorias:

- Políticas baseadas em papéis: são políticas que especificam as condições que um usuário deve possuir para torna-se membro de um papel. Tipicamente linguagens de política baseadas em papel, como Cassandra, *RT* e TPL, pertencem a esta categoria. As linguagens de políticas baseadas em papel relacionam usuários a seus papéis. O papel associado é, posteriormente, utilizado para decidir se o usuário pode ou não executar alguma ação.
- Políticas de controle de acesso: definem os pré-requisitos que um usuário deve cumprir para que tenha permissão de executar alguma ação. Estas políticas têm como objetivo limitar as atividades que o usuário pode executar.
- Políticas de privacidade: são políticas destinadas em proteger a privacidade do usuário. Elas devem refletir os regulamentos correntes de alguma empresa.
- Políticas de obrigação (*obligation policies*): especificam as ações que devem ser executadas quando algum evento ocorre, como, por exemplo, alguma violação de segurança. KAoS, Ponder e Rei são exemplos de linguagens de política que fazem parte desta categoria.

2.5. Linguagens de Regra na *Web* Semântica

Regra na *web* semântica é um meio de representar o conhecimento por meio de inferências que geralmente vai além da representação usada pela

linguagem OWL (*Web Ontology Language*) [Hebeler, 2009, p. 232]. Elas são tipicamente sentenças condicionais do tipo “se-então”. Quando um conjunto particular de sentenças for verdadeiro, novos conhecimentos serão adicionados na base de dados.

As regras na *web* semântica podem ser representadas via SWRL[1] (*Semantic Web Rule Language*), N3Logic [Berners-lee et al., 2008], SPIN[2] (SPARQL Inferencing Notation), entre outras linguagens. Tais linguagens são de propósitos geral, ou seja, não são destinadas especificamente para o controle de acesso.

2.5.1. SWRL

SWRL é uma linguagem de regras baseada em OWL DL e OWL Lite, e um subconjunto da linguagem RuleML[3] (*Rule Markup Language*), que modela cláusulas de Horn. O subconjunto de RuleML suportado por SWRL inclui somente predicados unário e binário. SWRL estende os axiomas OWL para incluir regras como *Cláusula de Horn*. Uma *Cláusula de Horn* representa a sentença condicional “se-então”, mais formalmente referida como implicação.

Uma regra em SWRL possui a forma de uma implicação entre um antecedente (*body*) e um consequente (*head*). O significado pretendido de uma regra em SWRL pode ser lido como: a qualquer hora em que as condições especificadas no antecedente sejam satisfeitas, então as condições especificadas no consequente devem ser satisfeitas também.

O antecedente e consequente da regra são formados de zero ou mais átomos. Um átomo consiste de qualquer predicativo unário, predicativo binário, igualdade, desigualdade, ou funções *built-ins* [Hebeler, 2009, p. 234]. As funções *built-ins* permitem a transformação dos dados e envolvem operações matemáticas, comparações, construções de URI, manipulação de *strings*, datas, tempo, listas, entre outros exemplos.

A linguagem SWRL possui três diferentes sintaxes: a sintaxe abstrata que é uma extensão da sintaxe abstrata da OWL[4], e duas sintaxes concretas, uma

[1] http://www.w3.org/Submission/SWRL/

[2] http://www.spinrdf.org/

[3] http://ruleml.org/

[4] http://www.w3.org/TR/2004/REC-owl-semantics-20040210/

baseada em XML e a outra baseada em RDF. SWRL tornou-se W3C *Member Submission* em 2004.

2.5.2. N3Logic

Outra forma de expressar regras na *web* semântica é usando N3Logic. N3Logic é uma lógica que permite que regras sejam integradas no modelo RDF[5] (*Resource Description Framework*) [Berners-lee et al., 2008; Berners-lee et al., 2000]. N3Logic usa a sintaxe Notation3[6] (ou N3), e estende o modelo RDF para incluir um conjunto de predicados (como o de implicação que será visto adiante), variáveis quantificadas, grafos aninhados, e *built-ins functions*.

Notation3 é uma linguagem que possui uma sintaxe equivalente ao RDF/XML, porém de forma mais compacta e legível. Regras em N3Logic são expressas da seguinte forma:

$$\{< \text{conjunto de sentenças} >\} \text{ log : implies } \{< \text{conjunto de sentenças} >\}.$$

onde o sujeito e objeto são fórmulas e o "log:implies" é um predicado especial que une fórmulas. O conjunto de sentenças dentro das chaves não são afirmações no modelo RDF. A regra acima diz que se a fórmula da esquerda for verdade, então implica na fórmula da direita. N3 permite escrever o predicado "log:implies" de uma forma mais simplificada através do símbolo de implicação "$\Rightarrow$". Portanto, escrever "log:implies" ou "$\Rightarrow$" significa a mesma coisa.

Uma formula em N3 permite também, além das declarações de sentenças, fazer declarações de variáveis. Existem dois tipos de variáveis em N3: a variável universal e a variável existencial. A variável com quantificador universal é declarada usando a sintaxe "@forAll: var.", ou somente "?var.", onde "var" é o nome da variável. Já a variável quantificada existencialmente é declarada usando a sintaxe "@forSome: var.", ou simplesmente "_ var.", onde "var" também é o nome da variável.

As regras em N3 são processadas usando a máquina de inferência CWM (*Closed World Machine*) [Berners-lee, 2000]. Segundo o artigo de Berners-lee et al. (2008), CWM é um processador de dados RDF de propósito geral para a *web* semântica. Ele é uma máquina de inferência com encadeamento para frente que

[5] http://www.w3.org/TR/rdf-primer/

[6] http://www.w3.org/TeamSubmission/2008/SUBM-n3-20080114/

pode ser usado para transformar, consultar, checar e filtrar a informação. CWM usa lógica de primeira ordem sem a negação clássica e faz parte do SWAP (*Semantic Web Application Platform*). Ele foi escrito em Python e tem código aberto sob a licença W3C *software license*.

Outro exemplo de máquina de inferência para regras escritas em N3Logic é o *Euler Proof Engine*[7] que é uma máquina de inferência com encadeamento para frente e para trás e fornece suporte a provas baseadas em lógica com detecções de caminhos eulerianos. A *Euler engine* possui algumas funções mais avançadas, tais como: *proof explanation*, construção de um possível modelo e *counter-model*, *matching* de regras, além de possuir também um extenso conjunto de *built-ins functions* [Vanel, 2011; Roo, 2011]. Euler foi implementado em Java e está sendo ativamente mantido e usado em produção.

2.6. SHDM

SHDM (*Semantic Hypermedia Design Method*) é um método para o projeto de aplicações hipermídia para a *web* semântica [Lima, 2003]. O método SHDM é uma evolução do método OOHDM (*Object Oriented Hypermedia Design Method*) através da introdução de formalismos e primitivas da *Web* Semântica. Ele permite o projeto de aplicações hipermídia baseadas em ontologias, descritas através de metadados.

O método OOHDM é um método baseado em modelos que utiliza técnicas de orientação a objetos para o design de aplicações *web* [Schwabe e Rossi, 1998]. Devido ao fato do método SHDM ser uma evolução do OOHDM, ele manteve os mesmos fundamentos e a mesma estrutura de fases de modelagem do método original.

O método SHDM é composto por seis atividades distintas essenciais para o desenvolvimento de aplicações *web*, sendo estas: Levantamento de Requisitos, Modelagem de Domínio, Projeto Comportamental, Projeto Navegacional, Projeto de Interface e Implementação. O processo se dá de forma iterativa e incremental, e, para cada fase, um modelo é produzido. O método SHDM será apresentado no capítulo 4.

[7] http://eulersharp.sourceforge.net/

2.7. Synth

O Synth é um ambiente de desenvolvimento que dá suporte à construção de aplicações modeladas segundo o método SHDM, fornecendo um conjunto de módulos capazes de receber como entrada os artefatos gerados na execução das etapas do método SHDM e produzir como saída uma aplicação hipermídia descrita por estes artefatos. Ele também dispõe de um ambiente de autoria que facilita a inserção e edição desses artefatos através de uma interface de formulários que pode ser executada em qualquer navegador de internet [Bomfim, 2011]. O Synth será apresentado no capítulo 5.

2.8. Trabalhos Relacionados

Diversos trabalhos relacionados à área de controle de acesso em aplicações da *web* semântica foram desenvolvidos. Entre eles podemos destacar alguns como segue a seguir.

Mühleisen et al. (2010) descrevem um sistema chamado PeLDS (*Policy-enabled Linked Data Server*) que permite o armazenamento de triplas RDF e fazer o controle de acesso da informação armazenada de forma que o usuário possa especificar quais elementos do seu grafo RDF podem ser publicados e quais usuários terão acesso a estes elementos. A autenticação do usuário no sistema é feita de forma descentralizada através do protocolo FOAF+SSL [Story et al., 2009].

As políticas de controle de acesso são expressas usando-se uma linguagem criada pelos próprios autores chamada de PsSF (*Prolog-style SWRL Format*) na qual é baseada no SWRL (*Semantic Web Rule Language*). O Pellet[8] foi usado como máquina de inferência para avaliar as políticas de acesso escritas nessa linguagem. Pellet é uma máquina de inferência para ontologias escritas em OWL DL, mas também dá suporte a regras em SWRL. Uma aplicação de exemplo que usa o sistema PeLDS foi desenvolvida e chama-se *Distributed Address Book*. Ela permite aos usuários adicionarem contatos na agenda de telefone e especificar visibilidades de cada item de dado armazenado. Por exemplo, um usuário pode definir seu numero de telefone

[8] http://clarkparsia.com/pellet/

como privado e somente visível para sua família. Essa aplicação foi escrita na linguagem PHP.

Hollenbach, Presbrey e Berners-Lee (2009) desenvolveram também um sistema de controle de acesso que permite fazer tanto a autenticação dos usuários quanto a autorização de forma descentralizada, ou seja, ele é adequado a estrutura da *web* semântica, pois não depende de um banco de dados centralizado. A finalidade desse sistema é permitir que os usuários criem, editem e compartilhem colaborativamente *linked data*, e que o acesso a esses dados possam ser controlados de forma descentralizada.

A autenticação é feita também através do protocolo FOAF+SSL e o mecanismo de controle de acesso usado é a ACL (*Access Control List*) na qual foi armazenada em um arquivo RDF. O *World Wide Web Consortium*[9] (W3C) utiliza um sistema de controle de acesso similar baseado em um arquivo RDF contendo uma ACL, porém tem a limitação com os mecanismos de autenticação e autorização, pois são armazenados por um banco de dados centralizado.

Os autores criaram dois módulos no *Apache HTTP Server*[10] (Apache) para implementar os mecanismos de autenticação e de autorização chamados mod_authn_webid e mod_authz_webid, respectivamente [Presbrey, 2009]. O módulo de autenticação implementa o protocolo FOAF+SSL enquanto que o módulo de autorização controla a ACL.

O sistema ACL desenvolvido por Hollenbach, Presbrey e Berners-Lee (2009) é baseado no vocabulário[11] descrito na especificação *Web Access Control*[12] no qual são fornecidos os termos usados pela ACL. *Web Access Control* é um sistema de controle de acesso através de ACL em que os usuários, grupos de usuários e os recursos são identificados por uma URI. O usuário é identificado especificamente por um WebID[13] que é usado pelo mecanismo de autenticação FOAF+SSL. Um FOAF+SSL WebID é uma URI usado para identificar de forma única uma pessoa e as informações associadas a ela.

A Figura 3 ilustra o vocabulário usado pela *Web Access Control*. A classe acl:Authorization representa a política de acesso definida pela ACL e suas propriedades. A propriedade acl:accessTo define um recurso protegido (gen:InformationResource) cujo acesso está sendo autorizado; e

[9] http://www.w3.org/

[10] http://httpd.apache.org/

[11] Basic Access Control Ontology, http://www.w3.org/ns/auth/acl#

[12] http://esw.w3.org/WebAccessControl

acl:accessToClass refere-se a uma classe de recursos. As propriedades acl:agent e acl:agentClass definem, respectivamente, um agente e uma classe de agentes (ex. qualquer individuo do tipo “foaf:Person”) cujo acesso foi concedido. A propriedade acl:defaultForNew define que a política de acesso será *default* para os novos recursos. A propriedade acl:mode define o modo de acesso que é dado para o um agente (acl:agent) ou uma classe de agentes (acl:agentClass). Três tipos de modos de acesso foram definidos: (i) acl:Read, (ii) acl:Write, e (iii) acl:Control. A classe acl:Control é um modo particular de escrita: um usuário que tem o modo de acesso de controle (acl:Control) para um arquivo, ele poderá editar a ACL relacionada a este arquivo.

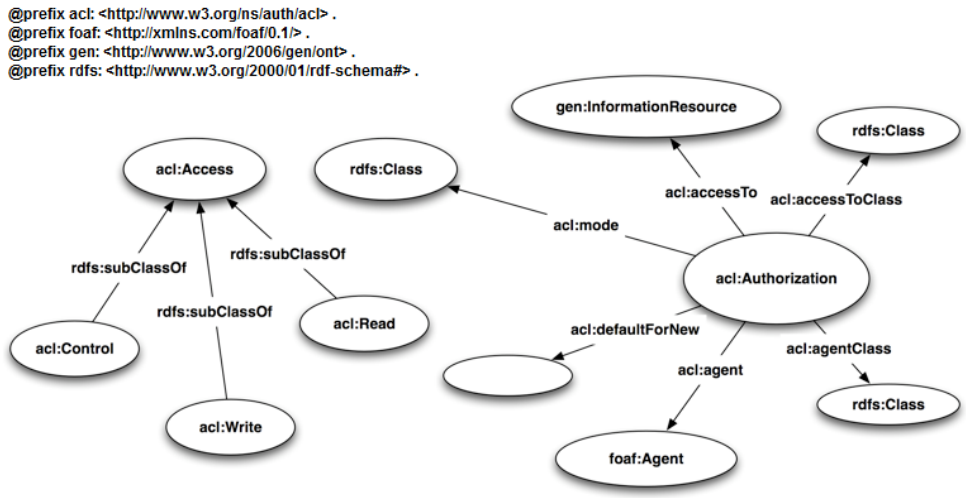

Figura 3 – Web Access Control (WAC) (Fonte: http://esw.w3.org/WebAccessControl/Vocabulary)

O Quadro 1 mostra um exemplo de uma política de acesso em ACL usando a notação *Notation3* ou (N3). Ela define que o usuário cuja URI é http://presbrey.mit.edu/foaf#presbrey tem direito de acesso a leitura, escrita e controle sobre o arquivo “file.rdf”, e que todos os outros usuários autenticados no sistema através do protocolo FOAF+SSL tem acesso de leitura a este arquivo. É definido também que os novos arquivos terão acesso permitidos de leitura, escrita e controle pelo usuário Presbrey. Se, por exemplo, um usuário autenticado pelo URI http://www.example.com/foaf#me tentar apagar o arquivo “file.rdf”, o sistema responderia com “403 Forbidden”.

[13] http://esw.w3.org/WebID

```
@prefix acl: <http://www.w3.org/ns/auth/acl#> .
[]
    a acl:Authorization ;
    acl:defaultForNew <.> ;
    acl:accessTo <file.rdf> ;
    acl:agent <http://presbrey.mit.edu/foaf#presbrey> ;
    acl:mode acl:Control, acl:Read, acl:Write .
[]
    a acl:Authorization ;
    acl:accessTo <file.rdf> ;
    acl:agentClass <http://xmlns.com/foaf/0.1/Agent> ;
    acl:mode acl:Read .
```

Quadro 1 – Exemplo de um arquivo ACL.

Uma interface *web* foi projetada para permitir que usuários autorizados possam visualizar e editar o arquivo RDF contendo a ACL diretamente via um navegador *web*. A autorização baseada em regras não é suportada. Portanto, políticas de controle de acesso mais sofisticadas como “somente os amigos de João e os amigos de seus amigos podem acessar os seus dados” só são possíveis através do uso de regras de controle de acesso. Porém, segundo os autores, é possível adicionar este recurso no sistema.

Finin et al. (2008) destacam que existem duas linhas de pesquisa sendo desenvolvidas de forma paralela sobre controle de acesso nos últimos anos. Uma linha está preocupada em desenvolver novos modelos de controle de acesso para atender as necessidades das políticas dos domínios de aplicações do mundo real, e, de forma paralela e quase independente, pesquisadores têm desenvolvido linguagens de política para o controle de acesso. Para os autores, os modelos e as linguagens para especificação de política de acesso devem ser aplicados de forma combinada. Um modelo sozinho pode não ser suficiente para expressar todos os detalhes de uma política de acesso, podendo deixar aspectos importantes não especificados. De forma contrária, uma linguagem de política sem estar relacionada a algum modelo, traz muita liberdade para o *designer* e pouca orientação para seu uso. O ideal seria poder ter modelos de controle de acesso suportados por linguagens de política de uma forma natural, e que os detalhes que não são capturados diretamente no modelo pudessem ser especificados pela linguagem de política.

Duas abordagens de como expressar os componentes e o comportamento do modelo RBAC usando a *Web Ontology Language* (OWL) [McGuinness, 2004] foram propostas. Embora a OWL não seja uma linguagem especificamente destinada para expressar políticas de autorização, ela possui um poder expressivo capaz de representar modelos como o do RBAC. Além disso, OWL é

um padrão usado pela W3C e já foi usada anteriormente para desenvolver linguagens de política para a *web,* tais como o Rei e KAoS.

Para cada abordagem, existe uma ontologia OWL que define os conceitos básicos do RBAC, incluindo notações tais como: sujeito, papel, objeto, ações e as associações feitas com o papel. Duas classes de ações foram incluídas para representar as ações permitidas e as ações proibidas. Além disso, cada uma das abordagens tem uma ontologia que modela o domínio específico da aplicação, definindo quem serão as classes de papéis, ações, sujeitos e objetos do domínio, as suas relações e atributos, e as ações permitidas e proibidas associadas a cada papel.

A primeira abordagem representa os papéis (rbac:Role) como classes e subclasses. A hierarquia de papéis do RBAC é representada pela hierarquia de classes OWL. Cada papel tem uma forma ativa (rbac:ActiveRole) associada a ele através da propriedade rbac:activeForm. A ação (rbac:Action) é representada por uma classe e tem exatamente um sujeito, no qual é uma instância da classe rbac:Subject, e um objeto que deve ser uma instância da classe rbac:Object. Uma ação pode ser tanto do tipo permitido (rbac:PermittedAction) quanto proibido (rbac:ProhibitedAction), mas não ambos. Propriedades importantes sobre o sujeito e/ou o objeto podem ser definidas na ontologia RBAC, enquanto que as propriedades adicionais são especificadas no modelo especifico do domínio. A Figura 4 mostra a ontologia OWL do modelo RBAC para esta abordagem.

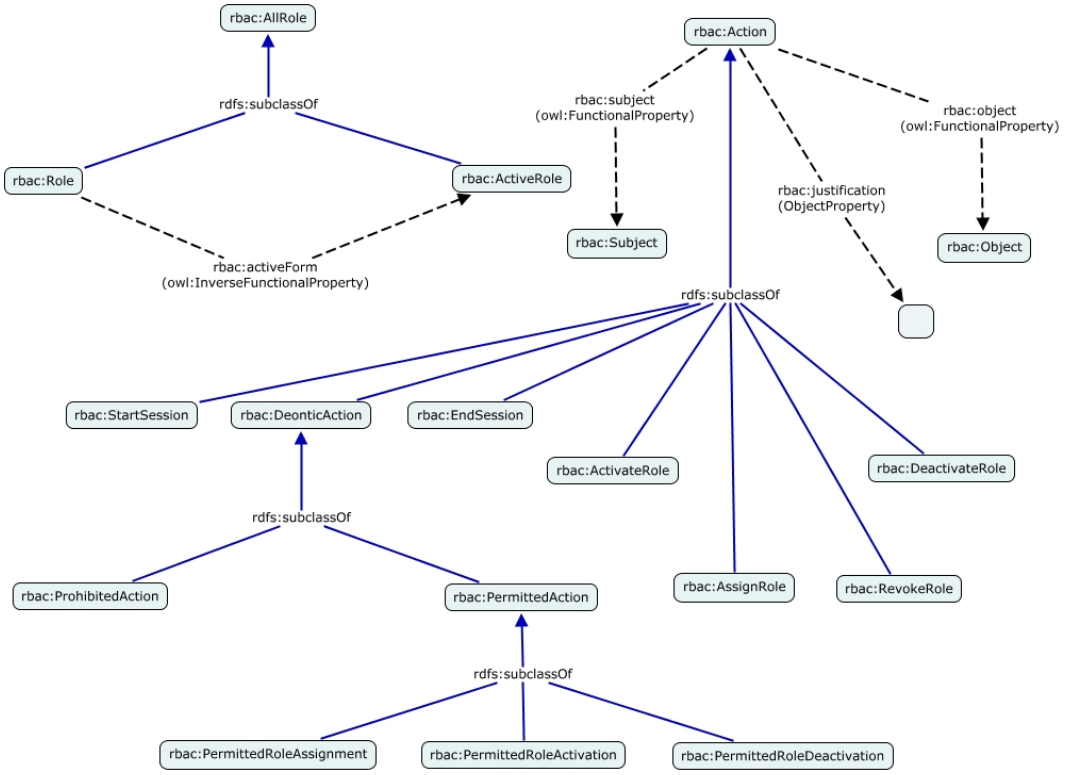

Figura 4 – Ontologia RBAC da primeira abordagem do RowlBAC

Para fazer a associação das permissões de acesso aos papéis, ou seja, definir as classes rbac:PermittedAction e rbac:ProhibitedAction, será necessário fazer uso de OWL Class Expressions[14] que são conceitos mais avançados baseados em lógica de descrição para a definição de classes OWL DL. O Quadro 2 mostra um exemplo de associação da permissão de imprimir ao papel ex:Faculty. Como somente os professores (ex:Faculty) podem imprimir na impressora dos professores, a ação :PermittedPrintOnFacultyPrinter foi criada como sendo uma subclasse de rbac:PermittedAction e que somente indivíduos que tiveram ativado o seu papel de ex:Faculty podem executar essa ação.

[14] http://www.w3.org/TR/2004/REC-owl-guide-20040210/#ComplexClasses

```
#Ações
:PrintOnFacultyPrinter a rbac:Action.

#Permissões
#Somente membros de Faculty tem a permissão de PrintOnFacultyPrinter
:PermittedPrintOnFacultyPrinter a rdfs:Class;
   rdfs:subClassOf rbac:PermittedAction;
   owl:equivalentClass [
      a owl:Class;
      owl:intersectionOf
        ( :PrintOnFacultyPrinter
            [ a owl:Restriction;
              owl:allValuesFrom ex:ActiveFaculty;
              owl:onProperty rbac:subject
            ]
        )
    ] .
```

Quadro 2 – Associação das permissões de acesso aos papéis

No entanto, segundo os autores, nem todos os componentes do RBAC puderam ser modelados usando a linguagem OWL. Para a definição das restrições estáticas e dinâmicas do modelo RABC e a ativação e desativação de um papel foi necessário o uso de uma linguagem de regras. Os autores usaram a linguagem N3Logic [Berners-lee et al., 2008] para a construção das regras. No entanto, SWRL[15] ou outras linguagens de regras poderiam ser usadas.

Na segunda abordagem, os papéis são representados como propriedades. Cada papel do domínio é uma instância da classe rbac:Role e possui duas propriedades: a propriedade rbac:role ligando o sujeito (rbac:Subject) aos seus possíveis papéis e a propriedade rbac:activeRole ligando o sujeito (rbac:Subject) aos seus papéis ativos. A hierarquia de papéis foi implementada através de regras usando a linguagem N3Logic. O Quadro 3 mostra como a hierarquia de papéis foi modelada nesta abordagem. A propriedade rbac:subRole determina que um papel é sub-papel de outro papel, ou seja, ele herda os privilégios do outro papel.

[15] http://www.w3.org/Submission/SWRL/

```
# role inheritance.
{?S role ?R.
 ?R subRole ?R2.
} => {?S role ?R2.}.

# activeRole inheritance.
{?S activeRole ?R.
 ?R subRole ?R2.
} => {?S activeRole ?R2.}.
```

Quadro 3 – Regra em N3Logic definindo a hierarquia de papéis

A associação das permissões de acesso com os papéis é feita pelas propriedades rbac:permitted e rbac:prohibited, que relaciona, respectivamente, um papel (rbac:Role) a uma ação permitida (rbac:PermittedAction), e um papel (rbac:Role) a uma ação proibida (rbac:ProhibitedAction). A Figura 5 mostra a ontologia OWL do modelo RBAC usando a segunda alternativa de modelagem do RowlBAC.

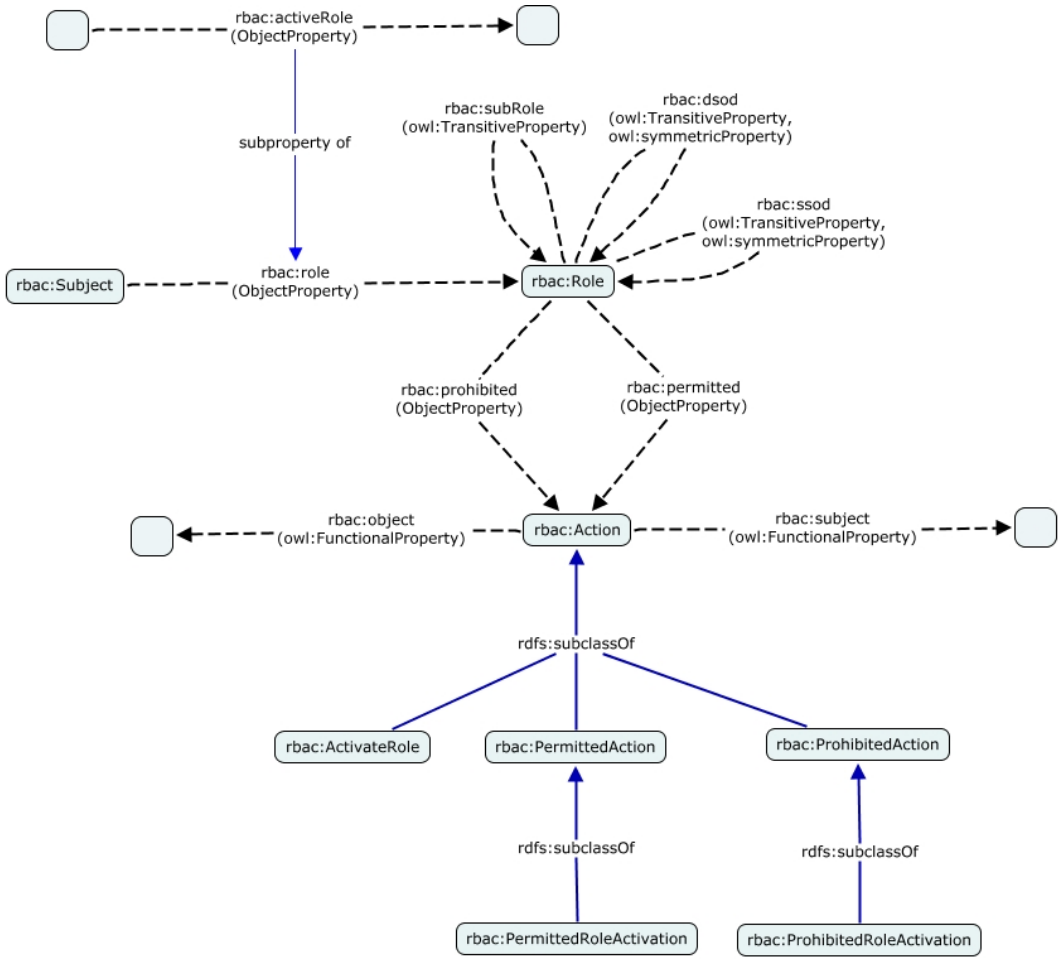

Figura 5 – Ontologia RBAC da segunda abordagem do RowlBAC

Como na abordagem anterior, segundos os autores, nem todos os componentes do modelo RBAC puderam ser especificados usando somente

OWL DL. Regras em N3Logic foram adicionadas na ontologia RBAC para definir a hierarquia de papéis (como visto acima), as restrições estáticas e dinâmicas do modelo RABC, a ativação e desativação de um papel e a associação das permissões com os papéis.

A proposta do RowlBAC de Finin et al. (2008) possui algumas limitações, tais como, na modelagem das restrições estáticas da primeira abordagem do RowlRAC é possível gerar uma inconsistência na ontologia [Ferrini at al., 2009]; algumas regras em N3Logic poderiam ser implementadas usando a própria lógica de descrição na qual a OWL se baseia. Outra limitação é que para cada avaliação da política de autorização é preciso executar a máquina de inferência, tornando as decisões de autorização custosas. Tais limitações foram superadas em outras propostas de descrever o modelo RBAC usando OWL, como as apresentas pelos autores Ferrini e Bertino (2009) e Knechtel e Hladik (2008). Tais propostas são descritas brevemente a seguir.

Ferrini e Bertino (2009) descreveram uma abordagem chamada de XACML+OWL que modela os componentes do modelo RBAC usando a linguagem OWL e as políticas de autorização usando a linguagem de política XACML (OASIS standard eXtensible Access Control Markup Language) [Moses, 2005] de forma integrada. A linguagem XACML possui uma extensão que suporta o modelo RBAC [Anderson, 2005], porém essa extensão não dá suporte a todos os seus componentes, como, por exemplo, as restrições estáticas e dinâmicas e a hierarquia de papéis. Para tanto, os autores estenderam a linguagem XACML e sua arquitetura de referência para permitir tais limitações, e integraram à sua arquitetura a capacidade de fazer inferências baseadas em OWL DL.

A representação do modelo RBAC usando ontologia OWL é mostrada na Figura 6. Os papéis de um domínio de interesse são representados por instâncias da classe *Role*. A hierarquia de papéis é definida pela propriedade subRoleOf(*Role, Role*), que é uma instância da classe owl:TransitiveProperty. É definida também a propriedade supRoleOf(*Role, Role*) que é o inverso (owl:inverseOf) de subRoleOf(*Role, Role*). A classe *Subject* representa o conjunto dos indivíduos que são sujeitos. A associação entre um sujeito com um papel é feita pela propriedade *hasRole*(*Subject, Role*). As classes *Resource* e *Action* representam os recursos e as ações, respectivamente. A classe *Permission* relaciona um usuário a uma ação-recurso. Esta relação é obtida através das propriedades *hasAction*(*Permission, Action*) e *hasResource*(*Permission, Resource*), ambas instâncias da classe

owl:FunctionalProperty. Por fim, a propriedade *hasPermission*(*Subject, Permission*) associa um sujeito a uma permissão de acesso.

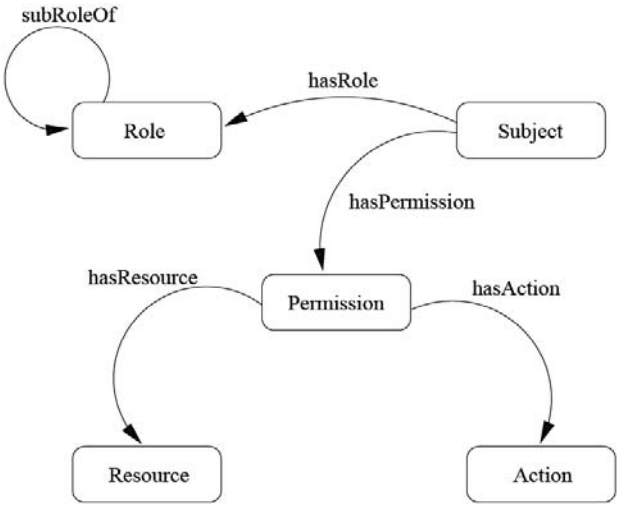

Figura 6 – Ontologia RBAC. (Fonte: Ferrini et al., 2009)

As restrições estáticas do modelo RBAC adicionam limites nas alocações dos papéis aos usuários, ou seja, para cada par de papéis com restrições estáticas, o usuário só poderá alocar um deles como seus possíveis papéis. As restrições estáticas foram modeladas adicionando-se os seguintes axiomas na ontologia OWL:

- $C' \equiv \exists hasRole : \{R_i\}.$
- $C'' \equiv \exists hasRole : \{R_j\}.$
- $C'\ owl : disjointWith\ C''$

As restrições dinâmicas impõem restrições na ativação do papel de um usuário, ou seja, para cada par de papéis com restrições dinâmicas, o usuário poderá ativar em tempo de execução somente um dos papéis. As restrições dinâmicas foram implementadas na ontologia OWL adicionando-se os seguintes axiomas:

- $D' \equiv \exists hasPerm\ .\ (\exists hasActi : \{R_i\} \cap \exists hasResorce : \{Resource\}).$
- $D'' \equiv \exists hasPerm\ .\ (\exists hasActi : \{R_j\} \cap \exists hasResorce : \{Resource\}).$
- $D'\ owl : disjointWith\ D''.$

Dois artigos de Martin Knechtel [Knechtel et al., 2008; Knechtel e Hladik, 2008] mostram uma abordagem para modelar uma extensão do modelo RBAC, chamado RBAC-CH, usando a linguagem OWL. RBAC-CH é um modelo de controle de acesso que estende o modelo RBAC adicionando uma hierarquia de classes de objeto. Sendo assim, a permissão de executar uma ação não será

associada a um objeto, mas sim a uma classe de objetos que poderá pertencer possivelmente a uma hierarquia de classes de objetos.

O modelo RBAC-CH foi todo implementado usando a lógica de descrição na forma de uma ontologia OWL 1.1. A linguagem OWL 1.1 é baseada na lógica de descrição SROIQ(D). Para a avaliação da política usando essa abordagem, a máquina de inferência baseada em OWL é executada uma única vez, e, em tempo de execução, as informações inferidas podem ser diretamente lidas para que a autorização possa ser decidida, podendo estas informações ser armazenadas em uma ACL. Os autores não levaram em consideração as restrições estáticas e dinâmicas do modelo RBAC.

Zhang et al. (2009) apresentam uma abordagem de como formalizar em lógica de descrição as políticas de controle de acesso mais comuns baseadas no modelo RelBAC (Relation Based Access Control) [Giunchiglia et al., 2008]. RelBAC é um modelo de controle de acesso projetado especialmente para atender os novos cenários da Web 2.0.