3
Modelo de Controle de Acesso no Projeto de Aplicações na Web Semântica

Este trabalho tem o objetivo de integrar o controle de acesso no projeto de aplicações na *web* semântica. Uma arquitetura de software para o controle de acesso foi projetada e implementada, e dois novos modelos foram adicionados. O primeiro descreve políticas de acesso em forma de regras, e o segundo provê a especificação de conceitos relacionados ao controle de acesso, tais como, sujeito, papel do sujeito, permissão, objeto e operação. O modelo RBAC (*Role-based Access Control model*) foi escolhido devido ao seu grande uso em aplicações do mundo real [Ferraiolo et al., 2007] e considerável estudo acadêmico [Finin et al., 2008].

Tais modelos podem ser aplicados em métodos de desenvolvimento de aplicações que tenham a noção de operação do domínio. Em especial, o método SHDM (*Semantic Hypermedia Design Method*) [Lima, 2003] foi estendido para a inclusão desses modelos de controle de acesso. O método SHDM e as mudanças ocorridas nele serão apresentados no capítulo 4.

A seguir será apresentado os modelos de controle de acesso utilizados neste trabalho.

3.1.
Modelo de Controle de Acesso

O modelo de controle de acesso implementado neste trabalho é baseado na segunda abordagem apresentada por Finin et al. (2008). Os autores desenvolveram duas abordagens diferentes para representar o modelo RBAC usando a linguagem OWL, incluindo os conceitos de sujeito, papel, objeto, ações e as associações feitas com o papel. A primeira abordagem representa os papéis como classes e subclasses, enquanto que na segunda abordagem, os papéis são representados como propriedades. Tais abordagens foram apresentadas na seção 2.8. Optou-se pela segunda abordagem, devido esta ser mais simples e concisa. A Figura 7 mostra o vocabulário do modelo de controle de acesso utilizado neste trabalho.

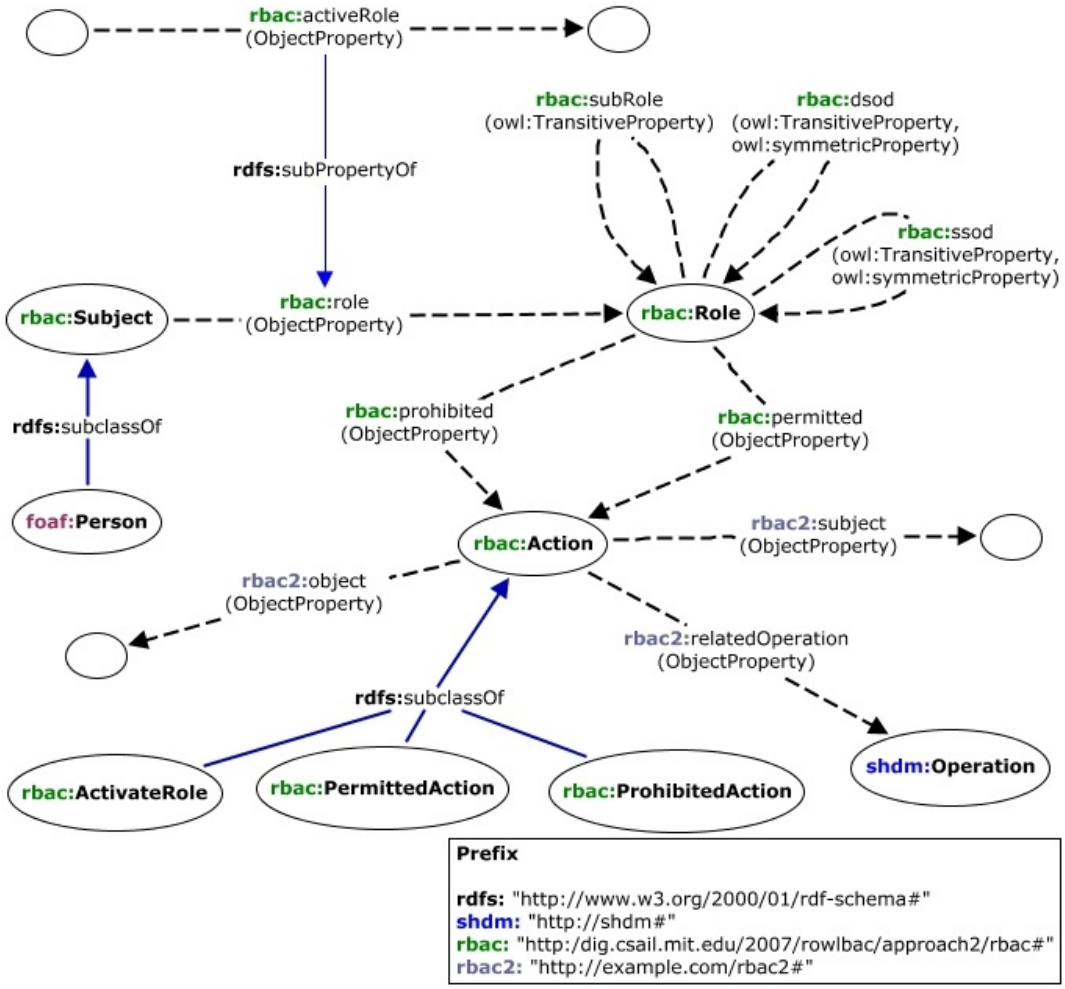

Figura 7 – Vocabulário do modelo de controle de acesso

Algumas modificações com relação ao modelo original foram feitas, como segue abaixo:

- A propriedade rbac:subject deixou de ser owl:FunctionalProperty, pois uma determinada ação (recurso da classe rbac:Action) pode ser realizada por mais de um usuário (recurso da classe rbac:Subject) que tenha permissão de realizar esta ação. Esta propriedade passou a ser rbac2:subject;
- A propriedade rbac:object deixou também de ser uma owl:FunctionalProperty, pois uma ação (instância da classe rbac:Action) pode estar relacionada com mais de um objeto (recurso da classe rbac:Object). Esta propriedade passou a ser rbac2:object;
- A propriedade rbac2:relatedOperation foi adicionada ao modelo para representar qual operação (shdm:Operation) definida no modelo de operações do método SHDM está sendo controlada;

- As classes rbac: PermittedRoleActivation e rbac: ProhibitedRoleActivation não são usadas por este modelo, por isso foram removidas;

3.1.1. Definição da Hierarquia de Papéis (rbac:subRole)

Os papéis são representados como instâncias da classe rbac:Role. A propriedade rbac:subRole especifica uma relação de herança entre dois papéis e é usada para criar uma hierarquia de papéis. Como visto na seção 2.2.2, a herança na hierarquia de papéis do modelo RBAC é dada da seguinte forma: todas as permissões definidas pelos papéis no topo da hierarquia são herdadas pelos papéis que estão na base da hierarquia. O Quadro 4 mostra um exemplo da definição de uma hierarquia de papéis para o domínio de um sistema de controle de submissões e revisões de trabalhos para conferências.

```
1  rbac:reviewer_role a rbac:Role .
2  rbac:senior_reviewer_role a rbac:Role ; rbac:subRole rbac:reviewer_role .
3  rbac:pcchair_role a rbac:Role ; rbac:subRole rbac:senior_reviewer_role .
4  rbac:author_role a rbac:Role .
5  rbac:conference_chair_role a rbac:Role .
```

Quadro 4 – Exemplo da definição de uma hierarquia de papéis

3.1.2. Definição dos Sujeitos (rbac:Subject)

A classe rbac:Subject representa os sujeitos de uma ação (recurso da classe rbac:Action). A propriedade rdfs:subClassOf relaciona uma classe que representa os usuários da aplicação (como, por exemplo, foaf:Person) do modelo de domínio com a classe rbac:Subject. A Quadro 5 mostra um exemplo da definição dos sujeitos.

```
1  foaf:Person rdfs:subClassOf rbac:Subject .
```

Quadro 5 – Exemplo da definição dos sujeitos (rbac:Subject)

3.1.3. Definição dos Objetos (rbac:Object)

Os recursos da classe rbac:Object representam as entidades do modelo de domínio que atuam como objeto de uma ação (recurso de rbac:Action). No domínio de um sistema de conferências, os artigos submetidos (swc:Paper) ou

as revisões de artigos (myconference:Review) podem ser considerados como objetos do modelo de controle de acesso, e podem ser definidos como mostra o Quadro 6.

```
1  swc:Paper rdfs:subClassOf rbac:Object .
2  myconference:Review rdfs:subClassOf rbac:Object .
```

Quadro 6 – Exemplo da definição dos objetos (rbac:Object)

3.1.4. Definição dos Possíveis Papéis (rbac:role)

A definição dos possíveis papéis que um usuário (recurso da classe rbac:Subject) pode ter é dada pela propriedade rbac:role. A propriedade rbac:role relaciona um sujeito (rbac:Subject) aos seus possíveis papéis, enquanto que a propriedade rbac:activeRole associa um sujeito (rbac:Subject) aos seus papéis ativos.

O exemplo apresentado no Quadro 7 define que um revisor (instância da classe myconference:Reviewer) pode ter como possíveis papéis o papel role:reviewer_role e o papel role:author_role, podendo então o revisor ativar quaisquer um deles ou ambos. Um revisor sênior (instância da classe myconference:SeniorReviewer), no exemplo do Quadro 7, pode ter somente o papel role:senior_reviewer_role. Como o papel role:senior_reviewer_role é rbac:subRole do papel role:reviewer_role, então um revisor sênior pode atuar também como um revisor. A definição dos possíveis papéis de cada sujeito é dada através de regras em N3Logic.

```
1  { ?R a myconference:Reviewer .
2  } => {?R rbac:role rbac:reviewer_role ; rbac:role rbac:author_role} .
3
4  { ?SR a myconference:SeniorReviewer .
5  } => {?SR rbac:role rbac:senior_reviewer_role} .
```

Quadro 7 – Exemplo da definição dos possíveis papéis para os sujeitos que são recursos das classes myconference:Reviewer e myconference:SeniorReviewer

3.1.5. Definição das Ações (rbac:Action)

As ações (rbac:Action), no modelo RBAC, são processos ativos que são invocados por um sujeito (recurso de rbac:Subject). O Quadro 8 apresenta

alguns exemplos de definições de ações para o domínio de um sistema de conferências.

```
1  CreateReview a rbac:Action .
2  EditReview a rbac:Action .
3  VisualizeStatusReview a rbac:Action .
4  CreatePaper a rbac:Action .
5  DownloadPaper a rbac:Action .
```

Quadro 8 – Exemplos da definição das ações (rbac:Action)

A propriedade rbac2:object relaciona uma ação (recurso da classe rbac:Action) a um ou mais objetos do domínio (recurso da classe rbac:Object). A definição dos objetos de uma ação é dada através de regras em N3Logic. O Quadro 9 mostra um exemplo da definição dos objetos da ação rbac:createReview. Tal definição especifica que a ação de criar uma revisão é aplicada somente sobre os recursos da classe swc:Paper. O Quadro 10 apresenta outro exemplo que define quais são os objetos para a ação rbac:editReview.

```
1  {   rbac:createReview a rbac:Action .
2      ?P a swc:Paper .
3  } => { rbac:createReview rbac2:object ?P } .
```

Quadro 9 – Exemplo da definição da propriedade rbac2:object para a ação rbac:createReview

```
1  {   rbac:editReview a rbac:Action .
2      ?REV a myconference:Review .
3  } => { rbac:editReview rbac2:object ?REV } .
```

Quadro 10 – Exemplo da definição da propriedade rbac2:object para a ação rbac:editReview

Algumas ações não possuem objeto. Alguns exemplos são: fazer *login*, criar um usuário na aplicação (rbac:createReviewer ou rbac:createPCChair), criar um artigo (data:createPaper), etc. Portanto, para estas ações, a propriedade rbac2:object não é criada.

3.1.6. Definição das Permissões (rbac:permitted)

A propriedade rbac:permitted é usada para definir quais ações (recurso de rbac:Action) são permitidas para um determinado papel (recurso de rbac:Role).

O Quadro 11 mostra um exemplo de definição de permissões para um sistema de conferências.

```
rbac:reviewer_role rbac:permitted rbac:createReview .
rbac:reviewer_role rbac:permitted rbac:editReview .
rbac:pcchair_role rbac:permitted rbac:createReviewer .
rbac:author_role rbac:permitted rbac:visualizeStatusReview .
rbac:author_role rbac:permitted rbac:createPaper .
rbac:conference_chair_role rbac:permitted rbac:createPCChair .
```

Quadro 11 – Exemplo da definição das permissões (rbac:permitted)

3.1.7. Regras em N3Logic Traduzidas em OWL 2

Algumas regras em N3Logic do modelo apresentado por Finin et al. (2008) foram traduzidas para a linguagem OWL 2. Por exemplo, para especificar a hierarquia de papéis, que antes foi implementada através de regras em N3Logic (Quadro 12), foi usado uma representação em OWL 2 de mesma semântica através da propriedade owl:propertyChainAxiom, como mostra o Quadro 13.

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

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

Quadro 12 – Definição da hierarquia de papéis usando regras em N3Logic

```
<ObjectProperty rdf:about="&Rbac;role">
    <rdfs:range rdf:resource="&Rbac;Role"/>
    <rdfs:domain rdf:resource="&Rbac;Subject"/>
    <rdfs:subPropertyOf rdf:resource="#topObjectProperty"/>
    <propertyChainAxiom rdf:parseType="Collection">
        <rdf:Description rdf:about="&Rbac;role"/>
        <rdf:Description rdf:about="&Rbac;subRole"/>
    </propertyChainAxiom>
</ObjectProperty>

<ObjectProperty rdf:about="&Rbac;activeRole">
    <rdfs:subPropertyOf rdf:resource="&Rbac;role"/>
    <propertyChainAxiom rdf:parseType="Collection">
        <rdf:Description rdf:about="&Rbac;activeRole"/>
        <rdf:Description rdf:about="&Rbac;subRole"/>
    </propertyChainAxiom>
</ObjectProperty>
```

Quadro 13 – Especificação em RDF das propriedades rbac:role, rbac:activeRole e da hierarquia de papéis usando a propriedade *Chain Axiom* da linguagem OWL 2

3.1.8. Definição da propriedade rbac2:relatedOperation

A propriedade rbac2:relatedOperation é usada para representar qual operação (shdm:Operation) definida no modelo de operações do método SHDM está sendo controlada. Essa propriedade tem uma relação inversa (owl:inverseOf) com a propriedade shdm:relatedAction. Esta última será definida no capítulo 4. O Quadro 14 mostra um exemplo da definição desta propriedade.

```
rbac:createReview rbac2:relatedOperation shdm:createReview .
```

Quadro 14 – Exemplo da definição da propriedade rbac2:relatedOperation

3.1.9. Execução do modelo RBAC

A execução do modelo RBAC para saber a permissão de execução de uma determinada ação (rbac:Action) foi feita através de regras em N3Logic seguindo a mesma abordagem de Finin et al. Uma pequena adaptação foi feita para lidar com o objeto das ações. A regra mostrada no Quadro 15, realiza a ativação do papel de um usuário. Tal definição permaneceu a mesma da original.

```
# role activation
{ ?A a rbac:ActivateRole;
     rbac2:subject ?S;
     rbac2:object ?R.
 ?S rbac:role ?R.
} => { ?A a rbac:PermittedRoleActivation; rbac:subject ?S; rbac:object ?R;
         rbac:justification "Is one of subjects possible roles".
       ?S rbac:activeRole ?R }.
```

Quadro 15 – Regra em N3 para a ativação do papel

A regra apresentada no Quadro 16 foi adicionada para verificar se uma ação (rbac:Action) possui um objeto (rbac:Object). Caso positivo, essa ação será do tipo rbac:ActionWithObject. For fim, o Quadro 17 mostra as regras em N3Logic que fazem a verificação se um sujeito tem permissão de executar uma determinada ação sobre certo objeto. Tal verificação foi dividida em duas regras (a) e (b) para tratar o caso de ações sem objeto.

```
# Check if action has object
{ ?A a ?RACTION ;
       rbac2:subject ?S ;
       rbac2:object ?O .

  ?RACTION a rbac:Action .
  ?S a rbac:Subject .
  ?O a rbac:Object .

} => { ?A a rbac:ActionWithObject } .
```

Quadro 16 – Regra em N3 para verificar se uma ação possui um objeto

```
# permission checking (1)
{ ?A a ?RACTION ;
       rbac2:subject ?S ;
       rbac2:object ?O .

  ?RACTION a rbac:Action .
  ?S a rbac:Subject .
  ?O a rbac:Object .

  ?ROLE rbac:permitted ?RACTION .
  ?S rbac:activeRole ?ROLE .

  ?RACTION rbac2:object ?O .

  ?A a rbac:ActionWithObject .

} => { ?A a rbac:PermittedAction ;
           rbac2:subject ?S ;
           rbac2:object ?O ;
           rbac:action ?RACTION } .
```

(a)

```
# permission checking (2)
@forAll F, S, O, A, RACTION, ROLE .
{ <Access_Control/ResultadoRegras_2.n3> log:semantics F .

  F log:includes {
                   A a RACTION ;
                     rbac2:subject S .

                   RACTION a rbac:Action .
                   S a rbac:Subject .

                   ROLE rbac:permitted RACTION .
                   S rbac:activeRole ROLE .
    } .

  F log:notIncludes { A a rbac:ActionWithObject } .

} => { A a rbac:PermittedAction ;
           rbac2:subject S ;
           rbac:action RACTION } .
```

(b)

Quadro 17 – Regras em N3 que decidem sobre a permissão de execução de uma ação

3.1.9.1. Perguntas de Autorização

Depois da definição de todas as diretivas do modelo de controle de acesso, consultas sobre a permissão de acesso podem ser criadas. Perguntas de Autorização foi o nome dado para as consultas sobre a permissão de execução de uma ação. O Quadro 18 mostra um exemplo para verificar se um usuário do modelo de domínio (Sebastian Rudolph) tem a permissão de executar a ação rbac:createReview atuando no papel de um revisor de artigos (rbac:reviewer_role). Caso positivo, essa ação será do tipo rbac:PermittedAction.

```
# SebastianRudolph activates his Reviewer role
rbac:sebastian_as_reviewer_role a rbac:ActivateRole ;
        rbac2:subject myconference:SebastianRudolph ;
        rbac2:object rbac:reviewer_role .

# Can SebastianRudolph CreateReview ? Yes, all Reviewer can CreateReview
rbac:sebastian_create_review a rbac:createReview ;
        rbac2:subject myconference:SebastianRudolph ;
        rbac2:object myconference:PaperF .
```

Quadro 18 – Exemplo de consulta de permissão de acesso

3.2. Modelo de Regras para as Políticas

De acordo com Latham (1985), uma política de acesso é um conjunto de regras. E 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 contextuais (como, por exemplo, o tempo).

Optou-se neste trabalho por usar a linguagem N3Logic [Berners-lee et al., 2008] para representar as políticas de autorização devido esta ter sido usada em outros trabalhos incluindo o trabalho previamente citado de Finin et al. O uso de linguagens de regras para a web semântica tais como, N3Logic e *Semantic Web Rule Language*[16] (SWRL), permite a criação de políticas de autorização mais complexas. Foi necessário representar tais políticas em um modelo de dados RDF.

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

A Figura 8 apresenta o modelo de regras usado para representar as políticas de controle de acesso.

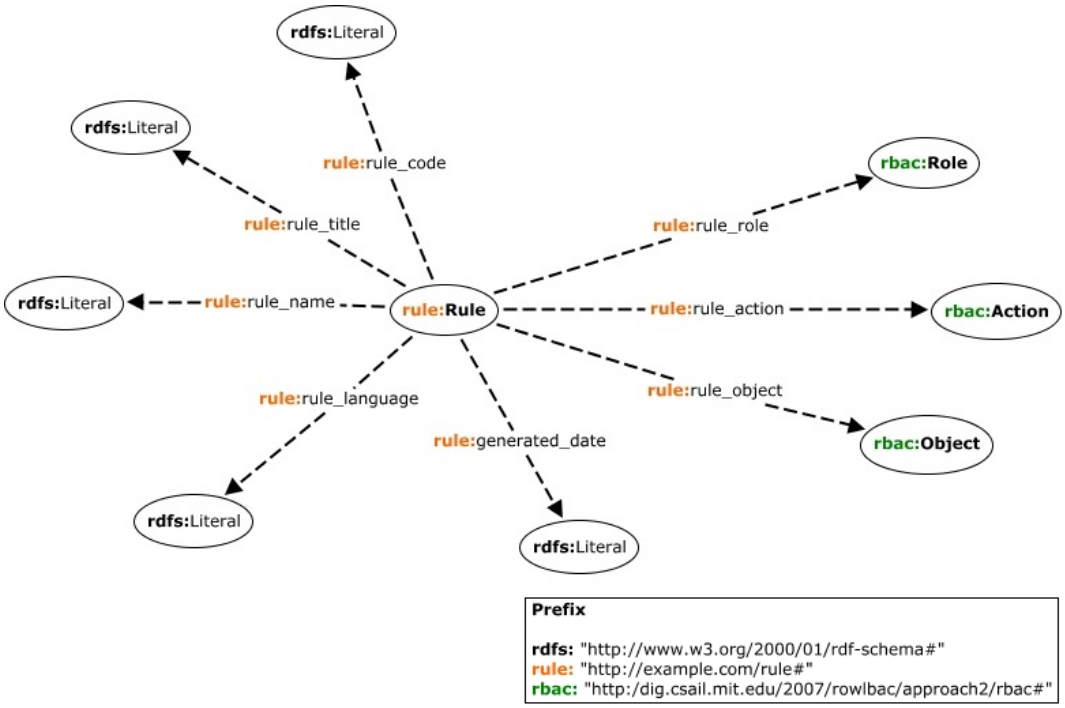

Figura 8 – Vocabulário do modelo de regras para as políticas

A classe rule:Rule representa as políticas de controle de acesso. Ela possui algumas propriedades do tipo *DatatypeProperty*, tais como

- rule:rule:name: define o nome da regra;
- rule:rule_title: define um título para a regra;
- rule:rule_code: define a especificação da regra;
- rule:rule_language: define a linguagem utilizada para descrever a especificação da regra;

Além destas, outras três propriedades foram definidas. São estas: as propriedades rule:rule_role, rule:rule_action e rule_object que são usadas para identificar, respectivamente, o papel (rbac:Role), a ação (rbac:Action) e o objeto da ação (rbac:Object) sobre os quais a regra se aplica. Tais propriedades vão ser úteis na etapa da geração das ACLs, que será vista com mais detalhes no capítulo 5.

O Quadro 19 mostra um exemplo de código de uma política de acesso que define que um revisor não pode revisar um artigo de um autor da mesma instituição que a dele. Outros exemplos serão apresentados no capítulo 6.

```
{   ?A a rbac:Action ;
      rbac:subject ?S ;
      rbac:object ?O .

    ?A a rbac:createReview .
    ?S rbac:activeRole rbac:reviewer_role .
    ?O a foaf:Document .

    ?S myconference:assigned_to ?O .

    ?AUTHOR rbac:role rbac:author_role .
    ?AUTHOR myconference:isAuthorOf ?O .

    ?AUTHOR myconference:memberOf ?I1 .
    ?S myconference:memberOf ?I2 .

    ?I1 log:uri ?URI1 .
    ?I2 log:uri ?URI2 .

    ?URI1 log:equalTo ?URI2 .

} => { ?A a rbac:ProhibitedAction } .
```

Quadro 19 – Exemplo de código de uma política de controle de acesso em N3Logic

3.3. Modelo de ACL da W3C

Uma Lista de Controle de Acesso (ACL) é um mecanismo rápido e simples de se implementar o controle de acesso em uma aplicação. Uma das desvantagens de uma aplicação baseada somente em regras é que o processo de autorização torna-se bastante custoso quando para cada decisão de autorização, uma nova execução da máquina de inferências de regras é realizada. Por conta disso, um modelo RDF para representar a ACL foi acrescentado e implementado de forma integrada com o modelo de controle de acesso baseado em papel apresentado na seção 3.1.

Desta forma, a máquina de inferência passará a ser executada apenas uma única vez e gerará como resultado uma ACL. Com isso, em tempo de execução as informações inferidas seriam diretamente lidas da ACL para que a autorização possa ser decidida. Sempre que houver alguma alteração nos dados, a máquina de inferência precisará ser re-executada e as permissões de acesso da ACL precisarão ser geradas novamente.

O modelo de ACL usado foi baseado no vocabulário de ACL da W3C[17]. O mesmo modelo também foi usado por Hollenbach, Presbrey e Berners-Lee (2009) para implementar um sistema de controle de acesso baseado em ACL. A Figura 9 ilustra o vocabulário do modelo de ACL usado neste trabalho. Tal modelo não sofreu alterações. A descrição dos termos deste modelo já foi apresentada na seção 2.8.

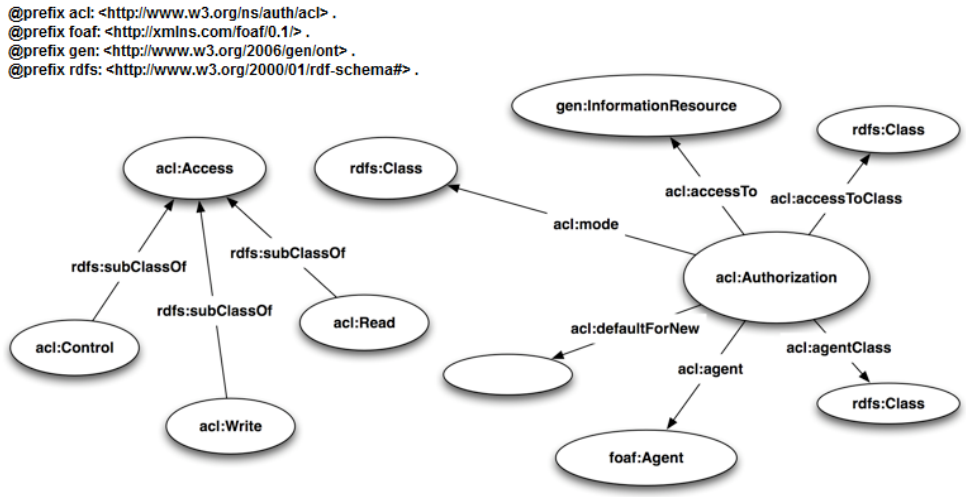

Figura 9 – Vocabulário do modelo de ACL (Fonte: http://esw.w3.org/WebAccessControl/Vocabulary)

3.4. Arquitetura de Software Modular para Controle de Acesso

Foi projetada uma arquitetura de software modular para o Sistema de Controle de Acesso proposto para esta dissertação, de forma que os componentes sejam independente de tecnologia de implementação e que haja baixo acoplamento entre eles. A arquitetura está dividida em duas partes: o mecanismo de autenticação que é responsável pela identificação do usuário no sistema, e o mecanismo de autorização que determina o que o usuário pode fazer no sistema. Esta arquitetura é composta por três módulos: o Módulo de Autenticação, o Módulo de Permissionamento e o Módulo de Inferência.

A Figura 10 mostra os componentes da arquitetura conceitual do Sistema de Controle de Acesso. As caixas de cor cinza com cantos arredondados representam os módulos e as caixas brancas representam os componentes da arquitetura. As setas não rotuladas têm o significado definido pela legenda correspondente.

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

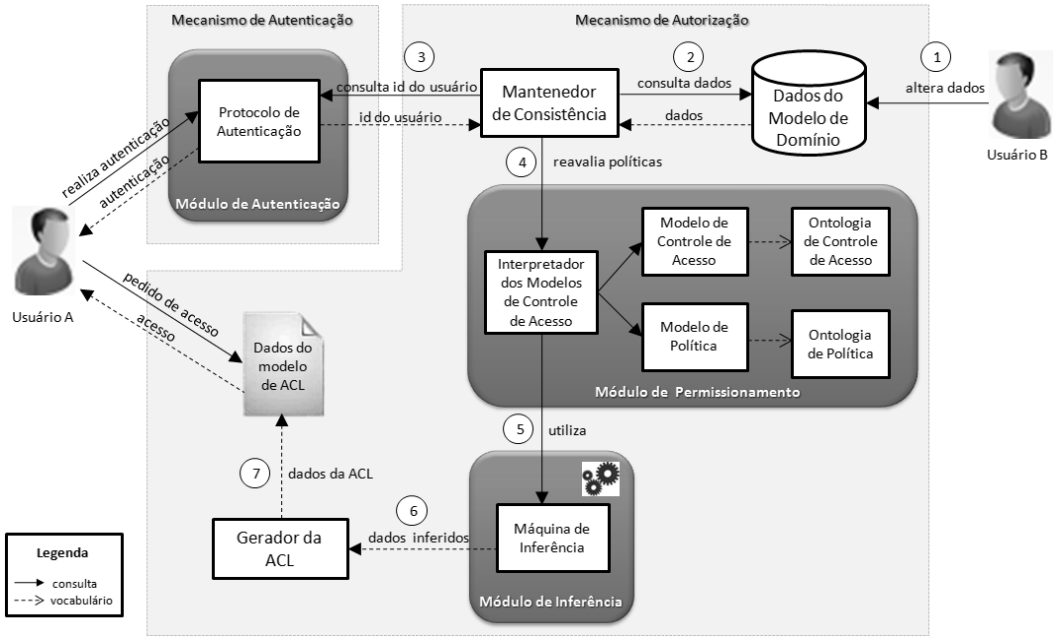

Figura 10 – Arquitetura Conceitual do Sistema de Controle de Acesso

O Módulo de Autenticação faz parte do mecanismo de autenticação e é responsável por executar um protocolo de autenticação, tal como o protocolo FOAF+SSL [Story et al., 2009] ou o protocolo OpenID[18].

O Módulo de Permissionamento pertence ao mecanismo de autorização e deve ser capaz de definir todas as diretivas do modelo de controle de acesso e as políticas de autorização da aplicação. Este módulo possui um Interpretador dos Modelos de Controle de Acesso cuja função é interpretar e manter as declarações sobre os modelos que definem o controle de acesso da aplicação. São eles: o Modelo de Controle de Acesso descrito de acordo com uma Ontologia de Controle de Acesso, e o Modelo de Política (de controle de acesso) descrito por uma Ontologia de Política.

O Módulo de Inferência possui uma Máquina de Inferência que é responsável por deduzir novos fatos a partir de fatos já existentes. As regras de controle de acesso são executadas neste módulo. Alguns exemplos são: Pellet[19] que é uma máquina de inferência para ontologias escritas em OWL DL, mas que também dá suporte a regras em SWRL; CWM (*Closed World Machine*) faz inferências de regras escritas em N3Logic; *Euler proof engine*[20] é uma máquina de inferência com encadeamento para frente e para trás e faz inferências de regras em N3Logic.

[18] http://openid.net/

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

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

3.5.
Arquitetura de Implementação

A Figura 11 mostra a arquitetura de implementação para o sistema de controle de acesso implementado neste trabalho.

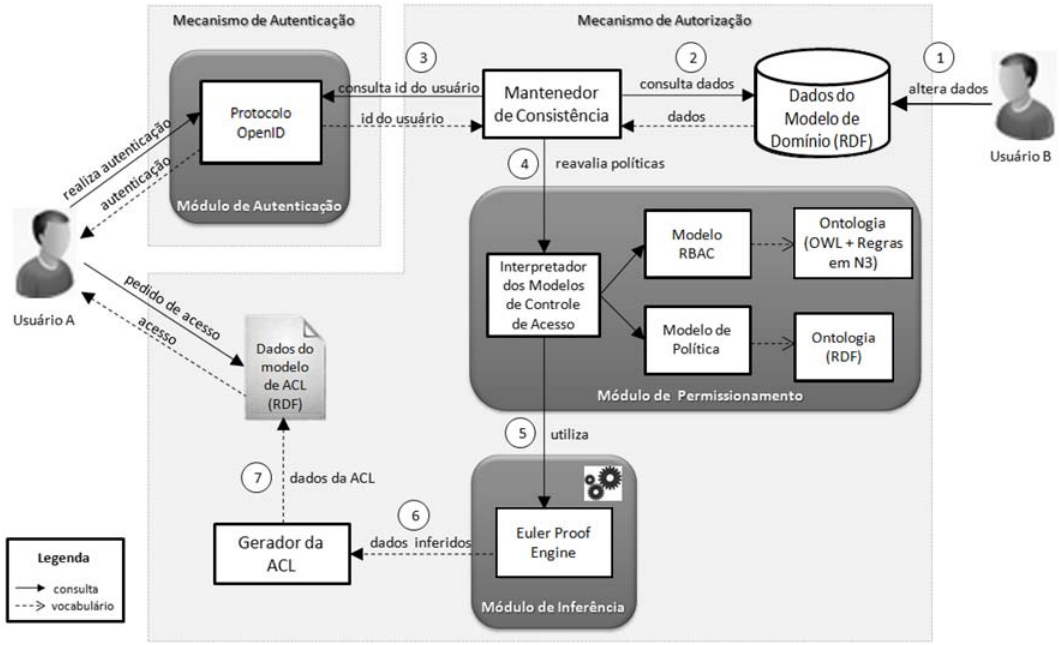

Figura 11 – Arquitetura de Implementação do Sistema de Controle de Acesso

O Protocolo de Autenticação implementado foi o protocolo OpenID [Recordon e Reed, 2006]. Tal protocolo foi escolhido devido ele ter suporte em serviços populares[21] tais como o MyOpenID[22], Google, Yahoo!, além de já ter sido usado em trabalhos passados como o trabalho de Cohen (2010). OpenID é um sistema distribuído de autenticação baseado em provedores de identidade. Em uma aplicação que aceite o OpenID, o usuário não precisa se cadastrar em tal aplicação, e sim por um provedor de autenticação que suporte o protocolo OpenID.

O Modelo de Controle de Acesso usado e a ontologia que descreve tal modelo foram apresentados na seção 3.1. O Modelo de Política usado e a ontologia que o descreve foram os mesmos apresentados na seção 3.2. A Máquina de inferência usada foi a *Euler proof engine*, devido esta ter sido implementada em Java e ter também apresentado melhor desempenho em

[21] http://openid.net/get-an-openid/

[22] MyOpenID – https://www.myopenid.com/

relação ao CWM. Além disso, como citado na seção 2.5.2, ela tem sido ativamente mantida e usada em produção [Vanel, 2011; Roo, 2011].

O Modelo de Domínio é um conjunto de recursos RDF. Ele será apresentado no capitulo 4. O Modelo de ACL usado foi o mesmo apresentado na seção 3.3.

3.6. Descrição do Sistema de Controle de Acesso

A seguir segue a descrição do sistema de controle de acesso cuja arquitetura foi apresentada na seção 3.4.

Para o usuário realizar a autenticação, ele deverá fornecer as suas credencias para o Protocolo de Autenticação implementado. Caso as informações passadas estejam corretas, o usuário estará autenticado e o URI que representa este usuário será armazenado na sessão da aplicação.

Depois de realizada a autenticação, o usuário poderá solicitar a permissão de acesso a algum recurso controlado pela aplicação. Para isso, o usuário deverá informar que operação este deseja executar e sobre qual objeto, caso exista. A aplicação irá verificar se a permissão está presente no modelo de ACL, e caso esteja, o usuário estará autorizado a executar a operação. Caso contrário, uma mensagem de erro deverá aparecer para o usuário.

A proposta do sistema de controle de acesso implementado neste trabalho foi de integrar o modelo de ACL com o modelo de controle de acesso baseado em papel junto com as políticas de acesso baseadas em regras. Desta forma, em tempo de execução, o usuário passa a acessar suas permissões de acesso diretamente na ACL, que foi gerada previamente por uma única execução da máquina de inferência.

No entanto, a ACL precisará ser atualizada toda vez que algum dado da aplicação seja alterado. Quando isso ocorre, o componente Mantenedor de Consistência da arquitetura será disparado. Ele é responsável por manter a consistência das permissões de acesso presentes na ACL. Portanto, o Mantenedor de Consistência recupera da sessão o identificador do usuário. Em seguida, ele aciona o Interpretador dos Modelos de Controle de Acesso a fim de que este consulte todas as políticas de controle de acesso e, logo após, mande executar a Maquina de Inferência para que as políticas (em forma de regras) contendo os dados alterados sejam executadas. For fim, a ACL é gerada novamente pelo componente Gerador da ACL a partir dos dados inferidos

(contendo as novas permissões de acesso) retornados pela Máquina de Inferência. Os passos da reavaliação das políticas de acesso estão numerados na arquitetura apresentada na seção 3.4.

3.7. Políticas Estáticas e Dinâmicas

Chamamos de Políticas Estáticas aquelas políticas de controle de acesso cujas permissões podem ser determinadas em tempo de compilação (geração da ACL). As Políticas Dinâmicas, no entanto, são aquelas cujas permissões não podem ser determinadas em tempo de compilação, isto é, elas dependem de alguma informação que somente é obtida em tempo de execução. Portanto, sempre que o usuário solicitar o acesso a algum recurso protegido da aplicação que dependa do resultado de alguma política dinâmica, então esta política precisará ser reavaliada em tempo de execução. Neste trabalho, somente as políticas estáticas foram implementadas.

A seguir segue alguns exemplos de políticas de acesso extraídos dos artigos da seção de trabalhos relacionados (2.8).

a) “Somente meus amigos e amigos de meus amigos podem acessar meus dados”;
b) “O usuário Horst tem permissão para acessar o número do telefone de Anna”;
c) “Os telefones de Horst somente são visíveis por seus familiares”;
d) “Os administradores de sistema tem acesso a todos os tipos de arquivos”;
e) “Todo professor pode submeter um projeto”;
f) “Professores associados podem revisar um projeto”;
g) “Professores podem usar qualquer impressora localizada em uma sala de aula”;
h) “Funcionários da universidade podem usar qualquer instrumento que esteja localizado em seu escritório”;
i) “O processo A só pode ser executado quando o processo B tiver sido inicializado”;
j) “Um professor pode revisar o mesmo projeto no máximo uma vez”;
k) “Um usuário não pode aprovar um projeto que ele mesmo submeteu”.

Tais políticas de acesso podem ser facilmente escritas na forma de regras. As políticas de a) a h) e a política k) são exemplos de Políticas Estáticas, pois as permissões de autorização de cada usuário podem ser obtidas em tempo de compilação, bastando executar as regras de controle de acesso uma única vez. Em particular, a política k) é estática, pois quando o usuário submete um projeto, ele estará mudando a base de dados para incluir este projeto. Portanto, quando o sistema de controle de acesso reavaliar tal política, a ACL não irá incluir para este usuário a permissão de aprovar seu próprio projeto. Consequentemente, ele não terá a permissão de fazer tal ação.

As políticas i) e j) são exemplos de Políticas Dinâmicas, visto que tais políticas dependem de alguma informação que só é obtida em tempo de execução. Na política j), o processo A depende da execução do processo B para que ele seja habilitado, e isso só poderá acontecer em tempo de execução. Na política j), a operação de revisar não envolve nenhuma mudança nos dados, e para verificar a quantidade de vezes que um usuário executou a operação de revisar, é preciso que a regra seja executada. É importante observar que se a operação de “revisar” implica em criar uma nova revisão, então essa política poderá ser considerada estática.