Instituto Superior de Engenharia do Porto

Framework de Controlo de Acessos para Triple-stores com interface Sail API

Raul Manuel Castro Valdoleiros Pinto de Oliveira

Dissertação para obtenção do Grau de Mestre em

Engenharia informática

Área de Especialização em

Tecnologias do Conhecimento e Decisão

Orientador: Nuno Silva

Júri:
Presidente:

Vogais:

Porto, Outubro de 2011

Resumo

Grande parte dos triples-stores são open source e desenvolvidos em Java, disponibilizando interfaces standards e privadas de acesso. A grande maioria destes sistemas não dispõe de mecanismos de controlo de acessos nativos, o que dificulta ou impossibilita a sua adopção em ambientes em que a segurança dos factos é importante (e.g. ambiente empresarial). Complementarmente observa-se que o modelo de controlo de acesso a triplos e em particular a triplos descritos por ontologias não está standardizado nem sequer estabilizado, havendo diversos modelos de descrição e algoritmos de avaliação de permissões de acesso.

O trabalho desenvolvido nesta tese/dissertação propõe um modelo e interface de controlo de acesso que permite e facilite a sua adopção por diferentes triple-stores já existentes e a integração dos triples-stores com outros sistemas já existentes na organização. Complementarmente, a plataforma de controlo de acesso não impõe qualquer modelo ou algoritmo de avaliação de permissões, mas pelo contrário permite a adopção de modelos e algoritmos distintos em função das necessidades ou desejos.

Finalmente demonstra-se a aplicabilidade e validade do modelo e interface propostos, através da sua implementação e adopção ao triple-store SwiftOWLIM já existente, que não dispõe de mecanismo de controlo de acessos nativo.

Abstract

The majority of triples-stores are open source and Java developed providing standard interfaces with private access. The biggest part of these systems doesn't have native access control mechanisms, making difficult or even impossible their adoption in environments where facts safety becomes important (e.g. business environment). In addition, it becomes visible that triples access control model, and particularly triples described by ontologies, is not standardized and not even stabilized, existing many description models and access permission evaluation algorithms.

This work proposes a model and an access control and interface that allows and facilitates his adoption by different existing triples-stores and their integration with other organization systems. In addition, the access control platform doesn't impose any model or permissions evaluation algorithm. In the other hand, it allows the adoption of distinct models and algorithms in according to needs or wishes.

Finally, is demonstrated the applicability and validity of the proposed model and interface through their implementation, and the adoption to the existing triple-store SwiftOWLIM that doesn't have a native access control mechanism.

Agradecimentos

Não pretendo fazer o mundo girar nem tenho essa pretensão, mas todos os dias fazem girar o meu. E as pessoas que o fazem girar merecem todo o meu carinho. Assim deixo o meu agradecimento à minha família e amigos, em especial à minha mãe e irmã.

Agradeço ao Senhor Engenheiro Nuno Silva por todo o tempo, empenho e paciência demonstrado comigo.

Índice

Lista de Figuras

Glossário

Nesta secção pretendem-se dissecar conceitos utilizados durante a escrita desta tese/dissertação.

ABAC, aka Attribute Based Access Control	Modelo de controlo de acessos baseado em atributos.
ALC	Sigla para Attributive Concept Language with Complements. Expressividade de ontologias, tem o nível mínimo de expressividade conhecida.
API	Sigla para Application Programming Interface, ou seja interface programática da aplicação.
Classificador	Componente capaz de inferir conhecimento a partir de factos de uma base de conhecimento descrita em OWL.
CRUD	Sigla para Create Retrieve Update Delete. É um padrão para a inserção, leitura, actualização e eliminação de factos num repositório.
Factos	Forma de representação de informação, podem ser descritos na forma de triplos ou noutra notação qualquer.
GUI	Sigla para Graphical User Interface, ou seja interface gráfica para o utilizador.
JAAS	Sigla para Java Authentication and Authorization Service. É a especificação standard para autenticação e autorização de aplicações Java.

Ontologia	Ontologia é a abstracção de linguagem de determinado ou determinados conceitos. É a representação da essência dos conceitos.
OWL	OWL ou Web Ontology Language foi desenvolvido direccionado para a Web para criar uma forma de tanto utilizador humanos como utilizadores maquina conseguirem ler e perceber conteúdos.
OWL API, Jena e Redland	Frameworks para processar RDF.
Papel, aka Role	Um papel identifica o tipo de utilizador, sendo que o utilizador pode ter vários papéis associados. Um papel é definido por um nome único e um conjunto de privilégios.
Privilégio	Um privilégio é o tipo base de autorização. É definido pelo par <objecto, nível/grau de acesso>, onde o Objecto pode ser uma classe ou o par <classe, propriedade>.
RBAC, aka Role Based Access Control	Modelo de controlo de acessos baseado em papéis.
RDF	RDF é a sigla para Resource Description Framework. É um standard para a representação e partilha de dados/factos na Web.
Sail API	API de baixo nível para acesso a repositórios RDF. Esta API é utilizada por vários triple-stores como o SwiftOWLIM, por exemplo.
Sesame	É uma framework para processar RDF.
SPARQL	É uma linguagem de pesquisa para RDF.
SwiftOWLIM	Repositório semântico baseado em triplos para acesso factos em RDF. Este triple-store tem a especificidade de ser também um plug-in para o Sesame.
Utilizador, aka Subject, sujeito, entidade	Um utilizador é uma entidade do sistema. Essa entidade consegue: Identificar-se, ou seja é capaz de efectuar autenticação; Ter papéis e/ou atributos associados; Consultar informação de um repositório RDF.

Capítulo 1

Introdução

É sabido que a privacidade é um dos aspectos mais relevantes da vida de qualquer ser humano. É inaceitável vivermos num mundo onde não consigamos manter os nossos segredos, escolher quem tem direito de os saber ou o que pretendemos que seja público. Facilmente imaginamos as repercussões caso os nossos pais/professores/chefes/amantes/etc. soubessem de tudo o que fazemos. Esta problemática é transposta para o mundo da informática, e em particular para uma das mais abertas, distribuídas e heterogéneas plataformas de recolha, armazenamento, partilha e processamento de informação existentes: a World Wide Web.

1.1 Web

A Web ou a World Wide Web tem passado por um processo evolutivo ao longo do tempo. Começou como sendo a Web 1.0 ou Web, evoluiu para a Web 2.0 ou Social Web e está em grande desenvolvimento a Web 3.0 ou Semantic Web, e a combinação da Web Social com a Web Semântica, dando origem à Web Social Semântica.

1.1.1 Web

Uma página diz-se que é puramente Web ou Web 1.0 quando não existe qualquer tipo de alteração de estado da aplicação no servidor por intermédio da interacção com o utilizador humano, e.g. quando a página apenas permite navegação.

1.1.2 Social Web

Uma página diz-se que é Social Web ou Web 2.0 quando é possível alterar o estado da aplicação no servido por intermédio da interacção com utilizador humano, e.g. quando é possível acrescentar comentários numa página. Este tipo de funcionalidade permite a comunicação entre utilizadores humanos, criando assim redes sociais. As denominadas redes sociais (e.g. Facebook[1], LinkedIn[2]) não são mais do que plataformas em que os utilizadores estabelecem explicitamente relações sociais (e.g. amizade, profissionais, interesses) e em que essas relações são aplicadas pelo motor de plataforma para sugerir e permitir o acesso a certos utilizadores à informação disponibilizada por outros utilizadores.

1.1.3 Semantic Web

Estamos perante a Web Semântica (Berners-Lee et al. 2001) quando utilizadores humanos e utilizadores máquinas lêem e compreendem sem ambiguidade o conteúdo das páginas. Isto tanto pode implicar que a página disponibilize em separado o conteúdo para ser humano (e.g. HTML) e para máquina (e.g. RDF[3]), como numa única variante (e.g. usando RDFa[4]). Para que seja também compreendida por computadores é preciso descrever o conhecimento utilizando semântica, semântica essa que é capturada e disponibilizada através de ontologias. A Web Semântica tem uma estrutura tecnológica vulgarmente representada num diagrama em pilha como o apresentado na Figura 1.

[1] http://www.facebook.com/

[2] http://www.linkedin.com/

[3] http://www.w3.org/RDF/

[4] http://www.w3.org/TR/xhtml-rdfa-primer/

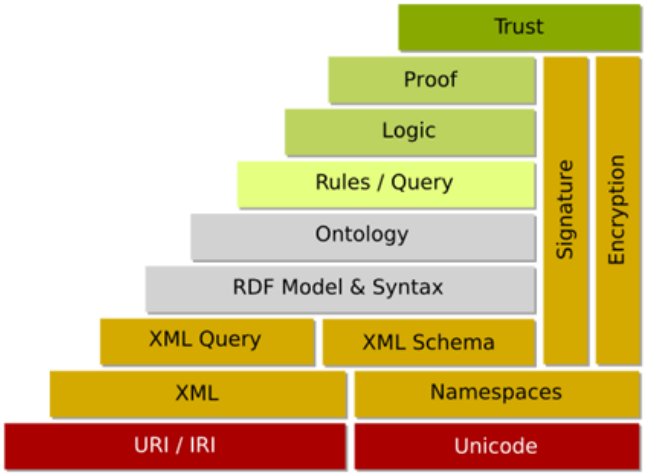


Figura 1 - Estrutura tecnológica da Web Semântica

Uma destas camadas diz respeito à confiança (Trust), pois sem confiança no controlo de acesso aos factos será difícil promover o uso de tal paradigma. É sobre esta dimensão que esta dissertação incidirá, ao propor uma abordagem tecnológica a incorporar nos repositórios de factos (vulgarmente denominados triple-stores ou quad-stores). Estes repositórios armazenam os factos modelados usando o modelo RDF, e que obedecem à semântica da ontologia (também ela muitas vezes representada em RDF). O mecanismo de serialização dos factos nos repositórios não é necessariamente XML, mas apenas na partilha e disponibilização entre sistemas (entre sítios web, entre sítios web e clientes).

1.2 Modelo de controlo de acesso

Um modelo de controlo de acesso é a definição de uma estrutura que possibilita controlar o acesso a recursos por parte de entidades, i.e. é o modelo que decide se a entidade tem ou não acesso a determinado recurso do sistema. Tal modelo é representado na Figura 2 usando nomenclatura UML relaxada.

Figura 2- Modelo de controlo de acesso

Existem essencialmente dois modelos de controlo de acesso:

- modelo de controlo de acesso baseado em atributos (ABAC)
- modelo de controlo de acesso baseado em papéis (RBAC).

1.2.1 ABAC

ABAC é o acrónimo para Attribute Based Access Control, i.e. controlo de acesso baseado em atributos. Isto significa que é pelos atributos que o modelo determina se o recurso pode ser acedido por determinada entidade. Os atributos podem ser aplicados a qualquer elemento do processo de controlo de acesso, ou seja os atributos podem ser aplicados a entidades, a recursos, ao próprio modelo, aos seus subelementos (Priebe et al. 2006).

Exemplificando, vamos supor um modelo de controlo de acessos para uma livraria. O modelo controla o acesso dos visitantes a livros da loja. As premissas para o modelo de controlo de acesso são as seguintes:

- Um livro só pode ser consultado por um visitante de cada vez;
- Se o visitante estiver na loja pela primeira vez tem prioridade na consulta dos livros;
- Um livro só pode ser consultado se já estiver em exposição.

Neste cenário o modelo de controlo de acessos podia definir um atributo "jaVisitou" que seria aplicado a todos os visitantes que já tivessem visitado a livraria, permitindo determinar a ordem de prioridades sobre as consultas dos livros. Sempre que alguém estivesse a consultar um livro seria preenchido o atributo "ocupado" do livro com o valor "sim" e quando alguém parasse de o consultar o valor passava para "não", o que permitia controlar a consulta do livro, à vez. Os livros que fossem requisitados para consulta teriam de ter o atributo "exposição" com o valor "sim". Desta forma o modelo de controlo de acesso só teria de interpretar os atributos e efectuar o controlo de acesso de acordo com as premissas.

1.2.2 RBAC

RBAC é acrónimo para Role Based Access Control, i.e. controlo de acesso baseado em papéis. Isto é, o acesso a um recurso é determinado pelo papel da entidade que acede ao recurso. O modelo de controlo de acesso cruza as permissões que estão associadas ao papel com o recurso que se pretende aceder, determinando assim se a entidade tem ou não permissões sobre o recurso.

Exemplificando, vamos supor um cenário de uma empresa que produz carros. É sabido que existem dois tipos de carros: aqueles em desenvolvimento e aqueles à venda. Os carros em desenvolvimento só podem ser vistos pelos construtores e os carros para venda podem ser vistos pelos construtores e pelos comerciais.

Para a resolução deste problema encaixa perfeitamente um modelo do tipo RBAC, ou seja são definidos dois tipos de recursos:

- CarrosDesenvolvimento;

- CarrosVenda.

E são também definidos dois papéis:

- Construtor;
- Comercial.

Define-se que o papel Construtor tem permissão de visualização sobre os recursos CarrosDesenvolvimento e CarrosVenda. O papel Comercial tem permissão de visualização sobre o recurso CarrosVenda. Isto significa que após o modelo de controlo de acesso ter identificado o papel da entidade consegue determinar de forma simples e intuitiva quais são os recursos aos quais a entidade tem acesso.

1.3 Serviços de controlo de acesso

Num qualquer processo de controlo de acesso existem diversas entidades (sistemas, sub-sistemas, etc.), cujos papéis no processo se limitam aos seguintes (Oracle Corporation 2008):

- **Policy Administration Point** (PAP) - tem como tarefa gerir as regras de acesso que um dado autor quer atribuir para os seus recursos.
- **Policy Decision Point** (PDP) - tem como principal tarefa verificar se um dado utilizador tem acesso a um determinado recurso com base nas regras definidas pelo autor do recurso no PAP.
- **Policy Enforcement Point** (PEP) - tem como principal tarefa aplicar a decisão tomada pelo PDP quando um utilizador tenta aceder a um determinado recurso.
- **Policy Information Point** (PIP) – tem como principal tarefa fornecer a informação necessária ao PDP, para que tome a decisão correcta.

Salienta-se que uma qualquer entidade envolvida no processo de controlo de acesso pode desempenhar um ou mais destes papéis simultaneamente.

1.4 Triple-stores

Os triple-stores são sistemas que permitem o armazenamento e manipulação de informação representada segundo o modelo de triplos introduzido pelo RDF. Triplos são uma forma de representar informação. É caracterizado por ser constituído por três elementos, dai o nome Triplo. É possível definir triplos com base noutros triplos, isto é utilizar um ou mais triplos na construção de outro. Exemplos de triplos:

- O João | come | a sopa;
- (O João | come | a sopa) | enquanto vê | televisão.

Quad-store é uma triple-store que permite o agrupamento de triplos em contextos. Deste modo, cada triplo é complementarmente caracterizado por um identificador que permite o seu agrupamento em contextos.

Um triple-store deve disponibilizar as seguintes funcionalidades:

- Parsing, de ficheiros externos em formatos não nativos, como XML e as várias variantes de serialização de RDF;
- Carregameento de factos externos. Esta é uma tarefa bastante exigente em termos de performance, e que é um dos mais usados indicadores de performance do triple-stores;
- Armazenamento, dos factos (ver secção 2.2)
- Inferência, usando para isso motores de inferência terceiros (e.g. Pellet, FaCT++);
- CRUD (Create, Reatrive, Update e Delete) de factos da base de factos.

Na sua grande maioria disponibilizam várias interfaces de manipulação de factos às aplicações, nomeadamente uma interface programática nativa, interface SPARQL (por HTTP e/ou programática), e uma interface comum (e.g. Sail, Jena). Esta interface abstrai a aplicação das particularidades do repositório lógico, que por sua vez é uma representação dos factos armazenados em mecanismos físicos específicos. Actualmente são habituais três tipos de armazenamento físico:

- Em memória, com todos os inconvenientes (persistência e limitações ao número de factos) e vantagens (performance) que daí advêm;
- Nativos, dizem respeito a mecanismos desenvolvidos especificamente para a manipulação de triplo, e.g. SwiftOWLIM[5];
- "Não memória, não nativo", nos quais se habitualmente se aplicam SGBD relacionais com esquemas optimizados para triplos, e.g. Jena SDB[6].

A Figura 3 representa uma arquitectura genérica de triple-stores, como descrito anteriormente.

[5] http://owlim.ontotext.com/display/OWLIMv35/SwiftOWLIM+Introduction

[6] http://openjena.org/SDB/

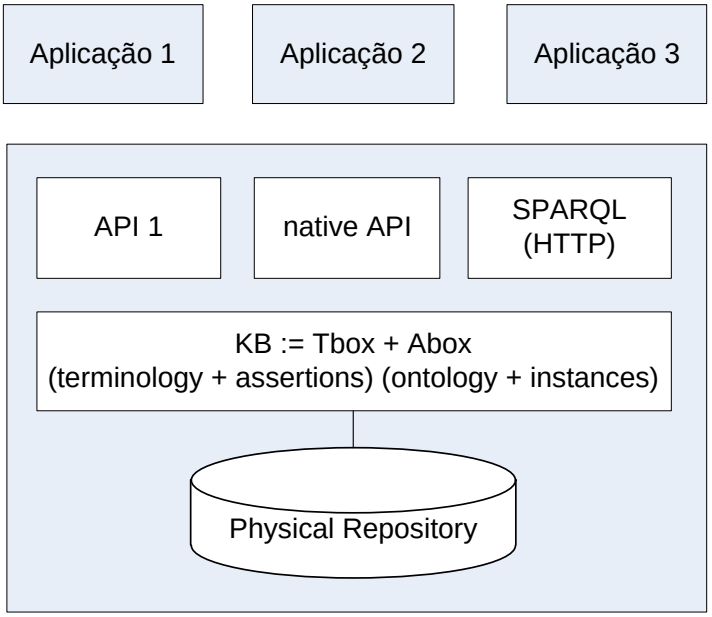


Figura 3 - Representação de arquitectura genérica de triple-stores

No contexto desta dissertação importa enumerar algumas limitações importantes dos triples-stores:

- Poucos dispõem de mecanismo de controlo de acessos, e os poucos que dispõem ou são mecanismos privados ou são dependentes do mecanismo de armazenamento SGBD relacional (usado vulgarmente como repositório físico);
- No caso dos mecanismos nativos, o modelo de controlo de acesso é único e não alterável;
- No caso dos mecanismos baseados no SGBD, o modelo de controlo de acesso não é adequado para triplos e ontologias;
- No caso dos mecanismos nativos, não permite a sua integração com ambientes onde autenticação é importante e já existente, nomeadamente com/em plataformas e aplicações empresariais.

De ora em diante a expressão "triple-store" é alternadamente substituída por "triple-store" ou "repositório de triplos" ou simplesmente "repositório".

Na secção 2.2 faz-se uma descrição de três triple-stores considerados parte do estado da arte.

1.5 Objectivos

Este projecto incide sobre o desenvolvimento de soluções que possibilitem ou promovam:

- Acesso controlado aos factos existentes no repositório de triple-stores descritos estruturalmente e semanticamente por ontologias;
- Integração de mecanismo de autenticação de triple-stores com ambientes distintos.

Pretende-se propor um modelo de plataforma para triple-stores que permita controlar o acesso aos factos na base de conhecimento e a sua fácil integração com ambientes distintos. O modelo tem de permitir definir o modelo de controlo de acessos a utilizar.

Depois da análise do contexto mais detalhadamente no Capítulo 2 será feito no Capítulo 3 um levantamento sistemático dos requisitos funcionais e não funcionais.

Capítulo 2

Contexto

Na primeira sub-secção deste capítulo descrevem-se quatro modelos/abordagens de controlo de acesso a repositórios de triplos descritos por ontologias. Na segunda sub-secção descrevem-se três triple-stores. Na terceira sub-secção descrevem-se três API de acesso a repositórios RDF.

2.1 Abordagens a controlo de acesso

Nesta secção faz-se uma descrição de quatro abordagens de controlo de acesso a repositórios progressivamente mais específicos

2.1.1 Role-Based Access Control Models

O artigo "Role-Based Access Control Models" (Sandhu & Sandhu, R., Coyne, E.J., Feinstein, H.L. and Youman, C.E. 1996) introduz o conceito de família de modelos RBAC de referência, no qual os papéis são definidos por um conjunto de permissões e um conjunto de utilizadores. Cada papel representa uma função dentro de determinado contexto, e.g. uma empresa, uma associação.

Modelos robustos de controlo de acessos baseados em papéis permitem relacionamentos entre os seus constituintes, i.e. entre papéis, permissões-papéis e utilizadores-papéis. Estes relacionamentos permitem determinar o acesso ou negação a determinado recurso do sistema.
Um modelo RBAC deve ter aproximadamente o mesmo peso para determinar quais os utilizadores de um papel e os privilégios desse mesmo papel. Deve também ter uma gestão centralizada de quais as permissões e utilizadores pertencentes a determinado papel.
Os autores do artigo definiram uma família de quatro modelos conceptuais, i.e. $RBAC_0$, $RBAC_1$, $RBAC_2$ e $RBAC_3$ (Figura 4).

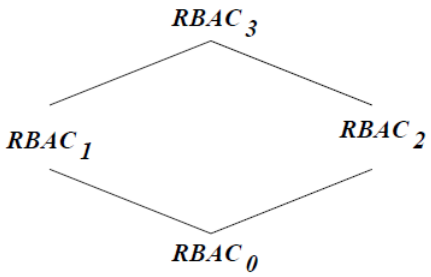


Figura 4 - Relação entre os modelos

O $RBAC_0$ é o modelo base, tem os requisitos mínimos para qualquer sistema que pretenda utilizar RBAC. $RBAC_1$ introduz o conceito de hierarquia de papéis, o $RBAC_2$ introduz o conceito de restrições. $RBAC_3$ é o modelo consolidado, incluindo os três anteriores.

2.1.1.1 Modelo base

O modelo base define os entre outros, os conceitos de utilizadores (U), papéis (R), permissões (P) e sessões (S). Um utilizador é sempre um ser humano. Um papel é uma função ou um título dentro de uma organização que confere autoridade e responsabilidade aos membros desse papel. Uma permissão é o consentimento de um determinado modo de acesso a um ou mais objectos (recursos) do sistema. As permissões são sempre na perspectiva de atribuir acesso e nunca de o negar. Na Figura 5 estão as relações de "user assignment" (UA) e "permission assignment" (PA). São relação de muitos-para-muitos em relação aos papéis, i.e. um papel pode ter várias permissões e uma permissão pode ter vários papéis. O mesmo acontece relativamente à relação entre os utilizadores e papéis.

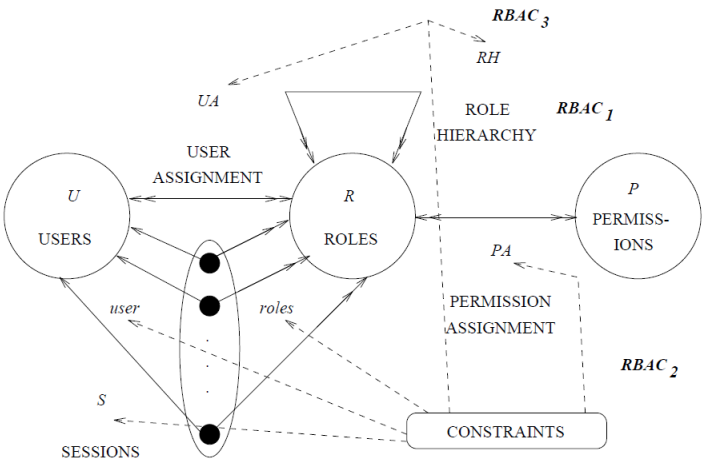


Figura 5 - Modelos RBAC

As sessões é o relacionamento de um utilizador para os possíveis papéis. Os papéis são activados consoante a utilização do sistema e ficam guardados no sistema em sessões. As permissões de um utilizador são a reunião das permissões dos papéis dum utilizador numa sessão.

Definição do modelo:

- U, R, P e S;
- PA ⊆ P x R, relação de muitos-para-muitos entre papéis e permissões;
- UA ⊆ U x R, relação de muitos-para-muitos entre papéis e utilizadores;
- Sessão de Utilizador: S → U, relação entre a sessão e o utilizador;
- Papéis por sessão: S → 2^R, uma relação entre cada sessão e um conjunto de papéis e permissões.

2.1.1.2 Hierarquia de papéis

Como demonstra já foi mencionado, o modelo $RBAC_1$ introduz a hierarquia de papéis (RH). Hierarquia de papéis é uma forma natural de organizar papéis. Os papéis de níveis superiores herdam as permissões dos papéis que lhe são inferiores, que se acumulam às permissões que já possuía. A Figura 6 apresenta um exemplo desta estrutura. Por exemplo, o papel "Project Supervisor" tem as permissões herdadas dos papéis "Test Engineer" e de "Programmer", bem como as permissões que lhe são atribuídas directamente.

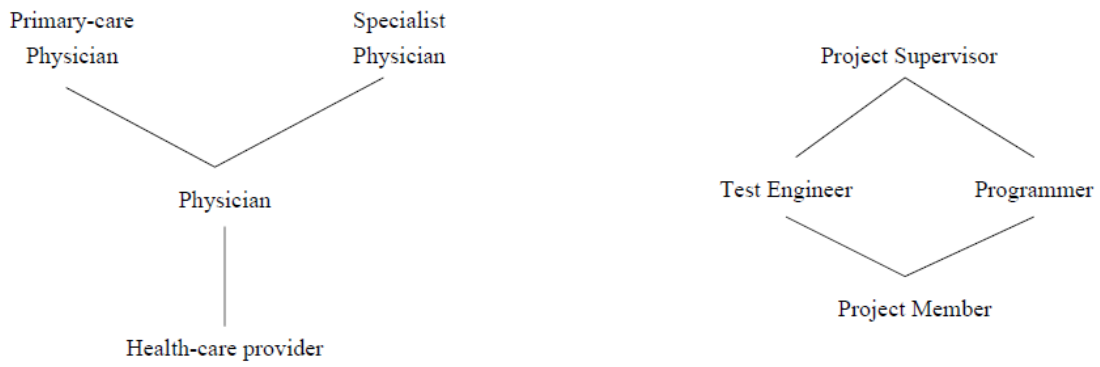


Figura 6 - Hierarquia de papéis

Matematicamente estas hierarquias são ordens parciais. Uma ordem parcial é uma relação reflexa, transitiva e anti-simétrica.

Definição do modelo:

- U, R, P, S, PA, UA, não sofrem alterações do modelo $RBAC_{0;}$
- RH ⊆ R x R é uma ordem parcial em R denominada hierarquia de papéis;
- Papéis: S → 2^R é modificado de $RBAC_0$ para forçar o uso de papéis.

Por vezes é necessário limitar o acesso a determinados recursos, e.g. um trabalho de programação inacabado deve apenas estar visível para os programadores e não para o supervisor do projecto. No entanto todos os trabalhos terminados devem estar visíveis para todos os membros do projecto. Para tal existe o conceito de papel privado, i.e. um papel que é atribuído apenas aos utilizadores específicos de determinador papel (Figura 7). Os papéis privados, para melhor identificação, têm sempre uma apóstrofe no final do nome. Por exemplo, apenas o papel Programmer' pode ver os trabalhos inacabados.

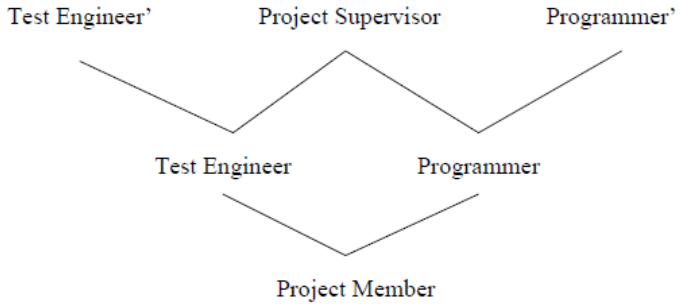


Figura 7 - Hierarquia de papéis com papéis privados

2.1.1.3 Restrições

Como já foi mencionado, o modelo $RBAC_2$ introduz o conceito de restrição. Restrições são um ponto de relevância dado que muitas vezes é argumentado como sendo a principal motivação do RBAC. Um exemplo de utilização de restrições é quando existem dois papéis que são

mutuamente exclusivos. Com restrições é possível garantir a integridade do sistema sem a supervisão de um perito de negócio.

As restrições podem ser aplicadas às relações UA e PA e também a utilizadores e papéis por várias sessões, tal como demonstra a Figura 5.

Definição do modelo: $RBAC_2$ é igual ao $RBAC_0$ exceptuando o facto de precisar de uma conjunto de restrições para determinar se as relações dos componentes são aceitáveis. As restrições devolvem os valores de "aceitável" ou "não aceitável".

Os autores definiram um conjunto de restrições comuns, que quando aplicadas têm implicações nas relações UA e PA, tais como todas as restrições:

- Restrição mutuamente exclusiva.
  - Em termos de UA é quando num conjunto de papéis um utilizador apenas pode estar atribuído a um desses papéis.
  - Em termos de PA é quando num conjunto de papéis uma permissão apenas pode estar atribuída a um desses papéis.
- Restrição de cardinalidade, quando se permite restringir o número mínimo ou máximo de utilizadores ou permissões.
- Restrição de pré-requisitos.
  - No contexto de UA, é quando um utilizador só pode pertencer a um papel se já for membro determinado papel;
  - No contexto de PA, é quando uma permissão só pode ser atribuída a um papel caso o papel já tenha determinada permissão.

Para que estas restrições funcionem é necessário que para cada ser humano exista apenas um utilizador no sistema e que para cada função exista apenas um papel no sistema.

2.1.1.4 Modelo consolidado

O modelo consolidado, $RBAC_3$, é a fusão de todos os modelos anteriores, como demonstra a Figura 4. A fusão dos dois modelos pode levantar questões importantes, onde pode ser necessário quebrar excepcionalmente alguns pressupostos dos modelos anteriores.

Supondo que uma empresa tem o modelo de papéis descrito na Figura 7 e os papéis de programador e engenheiro de testes são mutuamente exclusivos. Nessa empresa o modelo é aplicado a várias equipas mas numa delas não existem utilizadores suficientes para os papéis referidos terem pessoas distintas. Pode então o supervisor quebrar essa restrição, de forma que excepcionalmente seja ignorada a restrição desses papéis.

2.1.1.5 Modelos de gestão

A gestão de papéis nem sempre é centralizada no mesmo utilizador, tornando-se então necessário existir um modelo de gestão de papéis e respectivas permissões para que a integridade do sistema seja sempre garantida.

Como o objectivo principal de RBAC é a gestão de permissões, torna-se evidente que é possível utilizar um modelo de administração RBAC para gerir um modelo RBAC. Contudo, porque as permissões não podem ser aplicadas a elementos do modelo, apenas a dados ou recursos do sistema, torna-se necessário a existência de permissões e papéis especiais para manipular os conjuntos U, R, e P e as relações PA e UA: são denominadas permissões administrativas e papéis administrativos.

O modelo de gestão fornecido pelos autores está dividido em duas partes Figura 8. A parte de cima é idêntica ao modelo da Figura 5. A parte de baixo é o espelho da parte de cima, mas desta feita vocacionado para administração. De notar que as restrições podem ser aplicadas a ambas as partes da figura. É de realçar que nenhum elemento de uma parte comunica com um elemento de outra parte. Autoridade administrativa pode ser vista como a possibilidade de alterar atribuições de utilizadores e permissões, além das relações entre papéis.

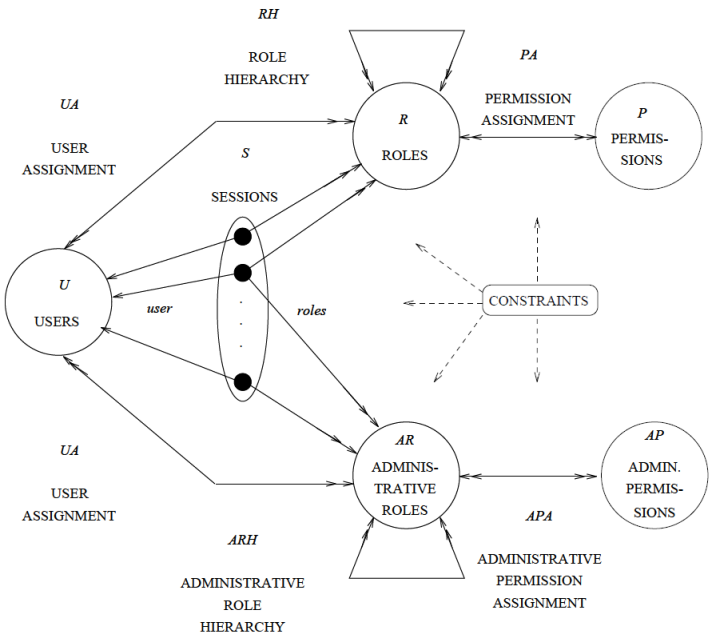
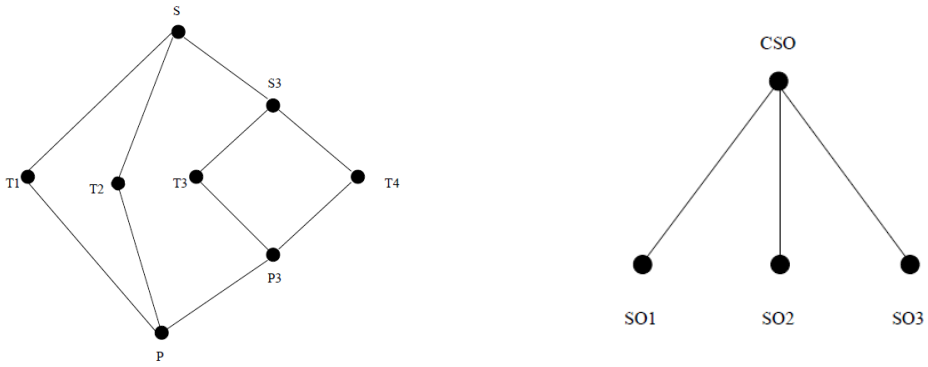


Figura 8 - Modelo de gestão RBAC

Um dos maiores problemas na gestão do modelo é como colocar o modelo de administração a gerir o modelo de papéis. Para isto, cada papel administrativo é responsável por gerir uma parte da hierarquia de papéis, e.g. o papel administrativo SO1 gere o papel T1 (Figura 9).



a) b)

Figura 9 - Hierarquia de papéis (a) e hierarquia de papéis administrativos (b)

No entanto, é necessário definir quais as permissões que os papéis administrativos detêm. Garantindo assim que os utilizadores dos papéis administrativos não executam tarefas que não lhe são devidas, pois gerir um papel não significa ter liberdade de acessos.

2.1.1.6 Conclusão

O artigo demonstra que RBAC pode ser usado em qualquer modelo (e.g. modelo de administração de papéis), não sendo preciso criar um modelo de segurança específico.
Os autores referem que é necessário um maior enfâse no estudo das restrições para que sejam criados normas de utilização. Os aspectos de gestão do modelo RBAC precisam de mais desenvolvimento, de forma a criar uma framework de gestão RBAC.

2.1.2 Administração de privilégios em RBAC

O artigo "Privilege administration for the role graph model" (Ionita & Osborn 2003) explica como é possível enquadrar um modelo de papéis num modelo de grafo de papéis para controlo de acessos baseado em papéis. Sendo assim o modelo de grafo de papéis pressupõe três planos:

- Grupos e utilizadores;
- Papéis;
- Privilégios.

2.1.2.1 Plano dos grupos e utilizadores

No plano dos grupos e utilizadores são definidos os conjuntos de utilizadores e as relações entre eles, i.e. são definidos os grupos (que são caracterizados por um nome e conjunto de utilizadores) e as relações entre os grupos (um grupo pode ser subgrupo de um outro grupo).

Exemplificando, podemos definir três grupos (Alunos, AlunosLicenciatura, AlunosMestrado). Definimos o grupo AlunosLicenciatura com todos os alunos da licenciatura e o grupo AlunosMestrado com todos os alunos do Mestrado. Agora podemos definir uma relação entre

eles, definindo que os grupos AlunosLicenciatura e AlunosMestrado fazem parte do grupo Alunos (Figura 10).

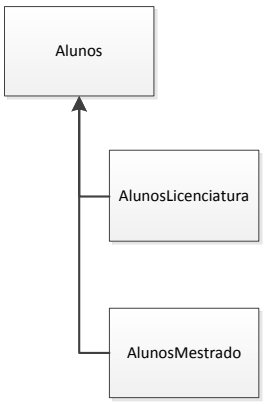


Figura 10 - Relação entre grupos

2.1.2.2 Planos dos papéis

No plano dos papéis é definida a hierarquia dos papéis, i.e. é possível definir-se que um papel (que é caracterizado por um nome e conjunto de privilégios) herda as características (conjunto de privilégios) de um outro papel.

Exemplificando, pode-se definir dois papéis, Aluno e Trabalhador-Estudante, onde se pretende que o papel de Aluno tenha os privilégios de um aluno normal e que o papel Trabalhador-Estudante tenha todos os privilégios de um aluno normal mais os privilégios de um Trabalhador-Estudante. Para isso definisse os privilégios do papel Aluno e faz-se o papel Trabalhador-Estudante estender do papel de Aluno adicionando-lhe os privilégios específicos dos trabalhadores-estudantes (Figura 11).

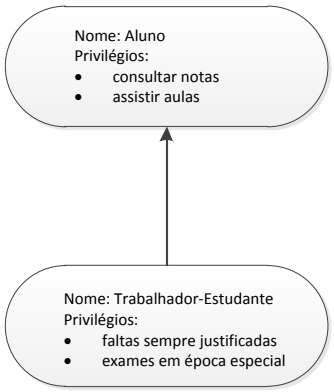


Figura 11 - Hierarquia de papéis

2.1.2.3 Plano dos privilégios

Um privilégio é definido pelo par <objecto, nível/grau de acesso>, sendo que o objecto é o recurso que se pretende controlar o acesso (e.g. facto, relação, comando) e o nível/grau de

acesso é a permissão (e.g. permite/não permite, leitura, escrita, remoção, criação) de acesso ao recurso..

Exemplificando, podemos definir um privilégio para o acesso as notas de um aluno e esse privilégio estará visível ao próprio aluno. O privilégio fica então definido da seguinte forma <nota, proprioAluno>.

2.1.2.4 Modelo do grafo de papéis

O modelo de grafo de papéis é um modelo para controlar o acesso a recursos, sendo este modelo baseado num grafo de papéis. O modelo de grafo de papéis é constituído por três grafos:

- grafo de papéis;
- grafo dos tipos de autorização;
- grafo de autorização de objectos.

O grafo de papéis é constituído por um papel que contém os privilégios máximos, denominado MaxRole, e por um papel com o mínimo de privilégios, denominado MinRole. É neste modelo onde são definidos os papéis e os privilégios correspondentes, bem como a hierarquia entre os papéis(Figura 12).

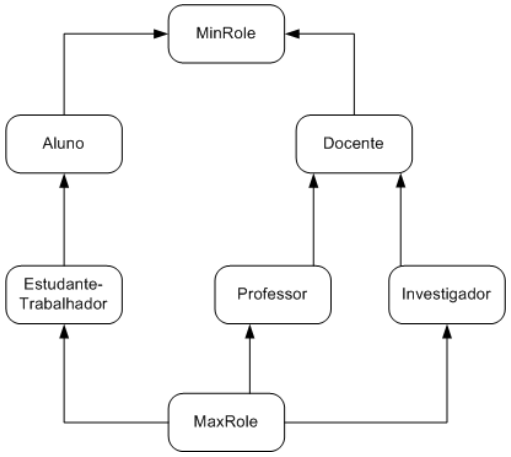


Figura 12 - Grafo de papéis

Os nós representam papéis e os nós directos/base são denominados is-junior. Há dois tipos de privilégios:

- Directos;
- Efectivos, que é a união dos privilégios directos e aqueles herdados dos papéis is-junior.

O grafo de papéis tem as seguintes características:

- Tem um único MaxRole;
- Tem um único MinRole;
- O grafo é acíclico, pelo que dois papéis não podem ter o mesmo privilégio

- Há um caminho entra o MinRole e todos os outros papéis;
- Há um caminho de qualquer papel até ao MaxRole;
- Se um papel contém os privilégios de outro role então tem de existir um caminho entre os dois papéis.

O grafo dos tipos de autorização (nível/grau de acesso) modela as implicações entre autorizações ou operações num sistema, i.e. quando a permissão de escrita no objecto O implica ter a permissão de leitura sobre esse objecto, esta relação denomina-se implies. A Figura 13 mostra um grafo que ilustra a relação implies entre vários níveis/graus de acesso a recursos.

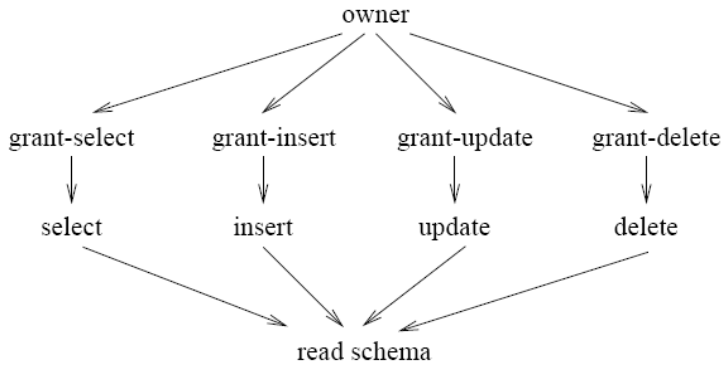


Figura 13 - Grafo dos tipos de autorização

O grafo de autorização de objectos modela as implicações que podem surgir devido ao agregar de objectos, i.e. quando um objecto agrega outros objectos é necessário que as permissões sobre os agregados também sejam contempladas. Por exemplo num modelo relacional a permissão de leitura de uma relação implica a possibilidade de ler qualquer índice dessa relação. As relações entre os objectos denominam-se contains. O grafo de autorização de objectos é composto por objectos individuais ou por contentores, onde os nós do grafo podem ter qualquer granularidade de objectos. A Figura 14 demonstra o grafo de autorização de objectos num modelo relacional.

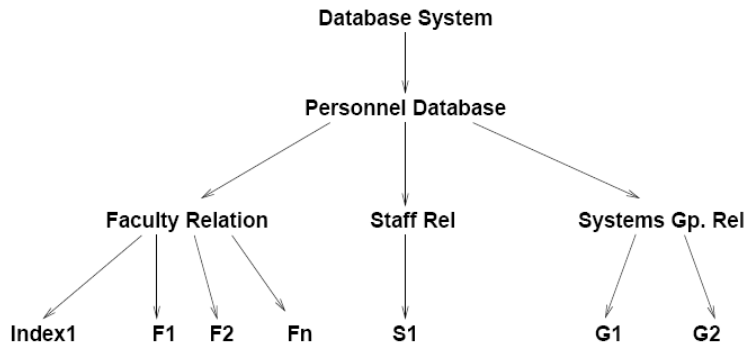


Figura 14 - Grafo de autorização de objectos

Existem três tipos de propagação que podem ser especificados sobre cada um dos dois últimos grafos mencionados para cada um dos elementos dos grafos de tipos de autorização e autorização de objectos:

- propagação para cima
- para baixo
- nula (não há propagação).

Exemplificando, utilizando o modelo relacional (Figura 13 e Figura 14) definimos que o tipo select propagará para baixo, isto é ter permissão de select numa "Faculty Relation" implica ter permissão de select em todos os seus objectos agregados (Index1, F1, F2 e Fn). Definimos agora que as permissões do grant-select não propagam, ou seja ter permissão de grant-select numa "Faculty Relation" não implica ter essa permissão nos seus objectos agregados. Se definirmos que o read-schema propagará para cima, então a permissão para ler o esquema da "Faculty Relation" implica ter permissão para ler o esquema de "Personnel Database".
A conjugação dos grafos tipos de autorização e autorização de objectos permite definir regras de propagação de permissões, é importante definir regras de propagação para se possa identificar quais os sub ou super recursos que o papel pode ver. É então introduzido o conceito de matriz de associação de autorizações, i.e. é uma matriz que indica os pares <Tipo de objecto, Tipo de autorização> que permite definir a propagação de permissões. Esta matriz contém todas as permissões de propagação, significando que a não existência de um par <Tipo de objecto, Tipo de autorização> significa a negação da propagação da permissão.

2.1.3 RBAC em repositories descritos por ontologias

O artigo "Specifying an Access Control Model for Ontologies for the Semantic Web" (Ionita & Osborn 2005) é uma adaptação do artigo descrito no capítulo anterior (Ionita & Osborn 2003) desta feita orientado para repositórios descritos por ontologias.

A Web Semântica é acedida por dois tipos de utilizadores através de ontologias: humanos e máquinas. A capacidade de inferência dos agentes de software é a maior preocupação de segurança, pois os agentes de software podem inferir conhecimento que não se queira divulgar, surgindo assim um problema de segurança.

A definição de papel é a mesma do artigo no qual este se baseia, i.e. são constituídos por um nome único e um conjunto de privilégios. Também a definição de privilégio é a mesma do artigo inicial, i.e. são constituídos pelo par <objecto, nível/grau de acesso>. Contudo, os tipos de acesso permitidos agora são (Figura 15):

- read, significa que sobre aquele objecto só pode ser feita leitura directa, não é permitida qualquer tipo de inferência ou navegação na hierarquia;

- reason, tem as características de read mas permite inferência sobre os conceitos e população da ontologia;
- extend, tem as características de read mas permite navegar na hierarquia das classes ou propriedades da ontologia.

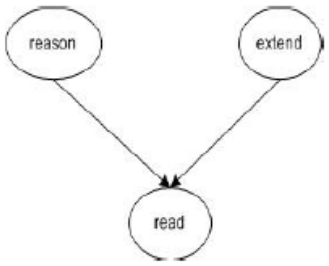
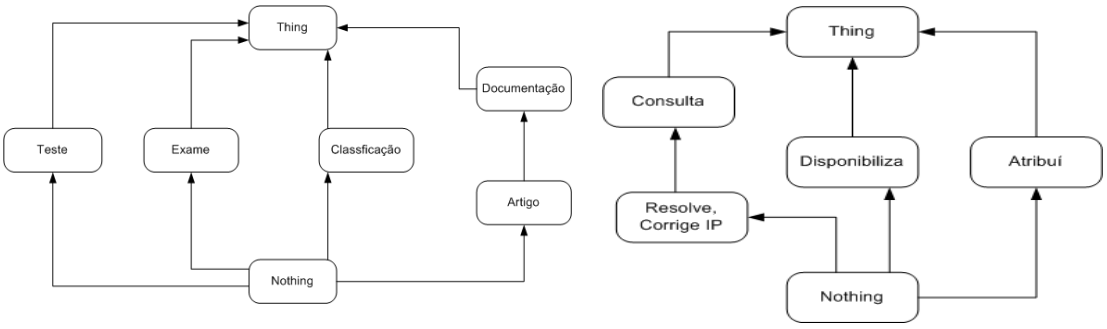


Figura 15 - Tipos de acesso

As classes e as propriedades são os objectos de autorização em OWL. Utilizando o grafo de papéis é necessário definir dois grafos, i.e. de classes e de propriedades, dado que os recursos são estruturados em grafos. Aqui o grafo de classes e propriedades da ontologia representam o grafo de autorização de objectos, como demonstra a Figura 16.



a) b)

Figura 16 - Grafo de classes e propriedades

Os grafos de classes e propriedades suportam algumas propriedades de OWL, i.e. podem ser aplicadas propriedades específicas a cada nó dos grafos. As propriedades suportadas por este modelo são:

- inverseOf;
- TransitiveProperty;
- SymmetricProperty;
- FunctionalProperty;
- InverseFunctionalProperty;

A propriedade "equivalentClass" pode apenas ser aplicada a nós do grafo de classes e significa que duas classes são equivalentes. Para se poder utilizar estas propriedades é necessário

acrescentar um indicador aos nós dos grafos. Assim é possível especificar um privilégio através do par <objecto, nível/grau de acesso> onde o objecto de autorização pode ser um ou ambos os recursos, i.e. classes e/ou propriedades. Na atribuição de privilégios o objecto do privilégio pode ser uma classe ou uma classe e propriedade. O padrão definido para a representação de um privilégio é <objecto>→<nível/grau de acesso>.

Considerando os grafos apresentados na Figura 16 a) e Figura 16 b), e ainda o da Figura 17, podemos definir que para o papel Professor:

1. Os exames corrigidos podem ser consultados segundo inferência;
2. A consulta dos artigos pode ser efectuada segundo navegação da hierarquia;
3. A documentação disponibilizada pode ser consultada apenas por acesso directo;
4. A atribuição da classificação pode ser consultada apenas por acesso directo.

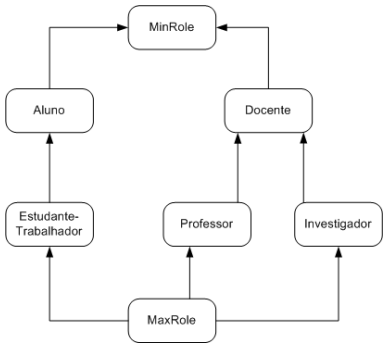


Figura 17 – Grafo de papéis

Utilizando a notação definida anteriormente fica que a definição dos privilégios tem os seguintes factos:

1. (Exame, Corrige) → Reason
2. (Artigo, Consulta) → Extend
3. (Documentação, Disponibiliza) → Read
4. (Classificação, Atribuí) → Read

Isto significa que se tivermos uma instância da classe Documentação podemos consultá-la dado que o privilégio nos permite navegar na hierarquia, i.e. como Artigo é subclasse de Documentação e o nível de acesso é extend é permitido que se deduza que Documentação é um Artigo permitindo acesso ao recurso. O mesmo raciocínio é efectuado para os restantes privilégios mas respeitando o seu nível/grau de acesso.

2.1.4 Combinação de RBAC e ABAC em Web semântica

O artigo "A Role and Attribute Based Access Control System Using Semantic Web Techonologies" (Cirio et al. 2007) demonstra como as tecnologias para a Web Semântica podem ser utilizadas para construir um sistema de controlo de acessos que combina o modelo de

papéis e de atributos. Como visto nas abordagens anteriores, a manutenção de utilizadores e papéis é efectuada apenas pelos administradores. Esta abordagem pode ser problemática quando se pretende sistemas mais dinâmicos. E.g. imagine-se que se deseja atribuir o papel de "primeiro cliente" a um cliente que entra pela primeira vez na loja, e nas restantes visitas ficar com o papel de "cliente". Para resolver este problema, o papel atribuído ao utilizador passa a ser escolhido com base num ou vários atributos, ao invés do papel atribuído pelo administrador ao próprio utilizador.

Os autores sistematizam a evolução do modelo referido em vários níveis:

- Modelo de acesso, com a introdução da identificação do papel ser com base em atributos;
- Framework para dar suporte à evolução do modelo de acesso recorrendo a tecnologia da Web Semântica, nomeadamente ontologias OWL e motores de inferência;
- SPARQL, como forma de ultrapassar as limitações de expressividade das ontologias OWL.

Foi desenvolvida a ontologia RBAC, deste artigo, com os seguintes princípios:

- A ontologia deve ser o mais abstracta possível;
- Não é tomado em consideração nenhum domínio de conhecimento;
- O modelo não tem nenhum fluxo de trabalho incluído.

A ontologia RBAC é utilizada para o desenho de outras ontologias, que a usam como base. No desenho das ontologias de domínio são tomados em consideração dois actores:

- o administrador de seguranças, responsável por definir privilégios e a relação entre privilégios e classes na ontologia de domínio
- o classificador DL, responsável por propagar os elementos de acordo com os axiomas da ontologia de domínio, através de inferência.

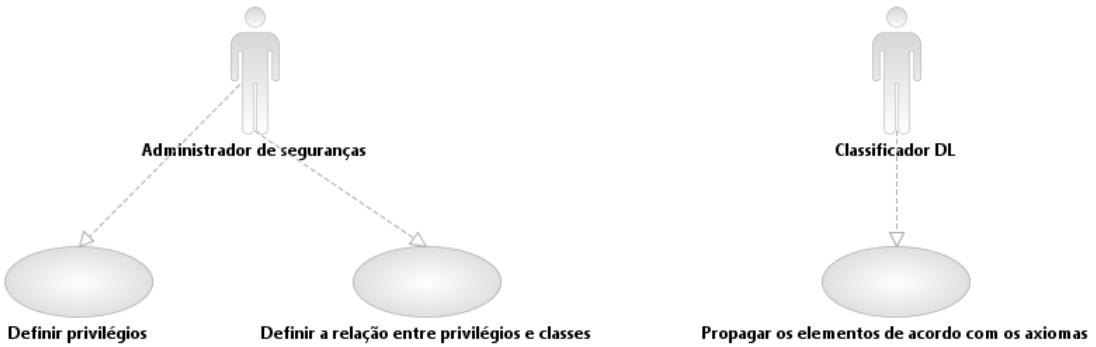


Figura 18 - Actores da ontologia RBAC

Na figura são apresentadas três ontologias, a ontologia RBAC e duas de domínio.

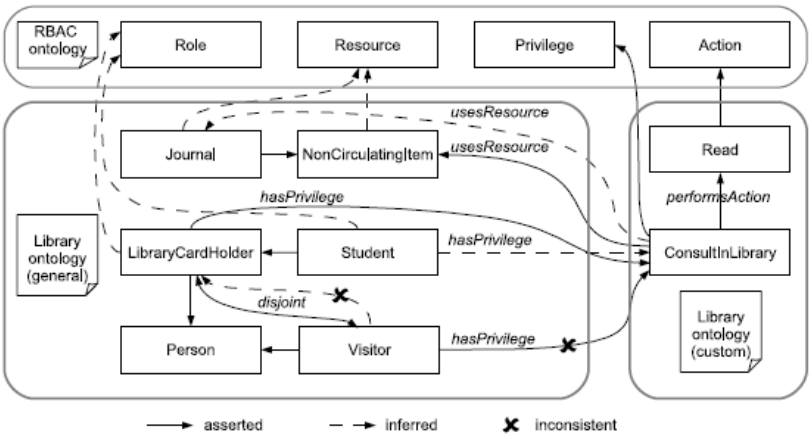


Figura 19 - Relacionamentos entre ontologias de domínio e a ontologia RBAC

Os autores referem que durante o desenvolvimento das ontologias utilizaram três padrões de design de ontologias:

- Value partition;
- Linked list;
- Recursive composition.

A ontologia RBAC tem quatro classes. Action é uma classe parcial que representa uma acção que um utilizador pode executar sobre um recurso, e.g. Read. Resource é uma classe que representa os objectos que se pretender aceder, e.g. Jounal. Privilege é uma classe parcial que representa o par (a,r) sendo que $a \in \text{Action}$ e $r \in \text{Relation}$, e.g. privilégio ConsultInLibrary representa o par (Journal,Read). Role é uma classe definida pela propriedade hasPrivilege cujo contra-domínio é Privilege, e.g. Student.

A ontologia RBAC tem as seguintes propriedades:

- hasPrivilege $\subset$ Role x Privilege, é uma relação de muitos-para-muitos entre papéis e privilégios;
- notTogetherWith $\subset$ Role x Role, é uma relação de muitos-para-muitos que evita o carregamento de papéis se outro determinado papel já estive carregado;
- performsActions: Privilege → Action, associada uma acção a cada privilégio;
- usesResources: Privilege → Resource, associa o privilégio ao recurso sobre o qual trabalha. A junção das duas últimas propriedades permite definir o par (a,r).

Devido ao uso de OWL, e de acordo com o explicado até ao momento, torna-se impossível definir que um estudante faz uma tese e essa tese é corrigida pelo seu orientador. Apenas é possível definir que um estudante faz uma tese e que um orientador corrige uma tese.

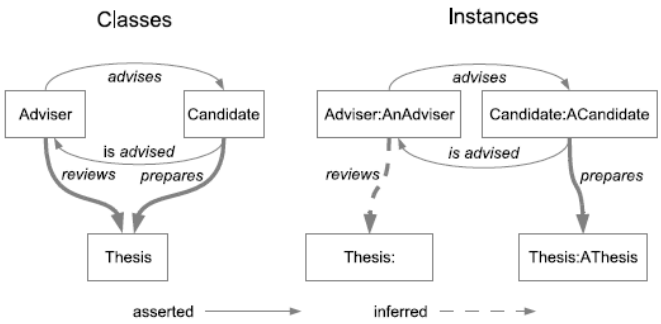


Figura 20 - Correcção de tese

Para ultrapassar esta limitação, os autores introduziram a aplicação de SPARQL à solução. Podendo, através da pesquisa, limitar o âmbito de execução, i.e. se o modelo considerar apenas a tese escrita pelo aluno como a única existente nesse contexto, então é essa tese que o orientador vai corrigir.

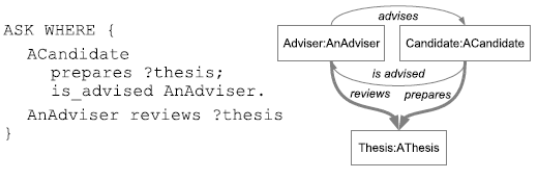


Figura 21 - Correcção de tese com âmbito de execução

O sistema construído assume vários pressupostos. O conhecimento utilizado para tomar decisões de acesso ou negação está descrito em ontologias OWL. A organização tem forma de registar, propagar, pedir e verificar os atributos dos utilizadores. Os atributos estão descritos sob o formato da ontologia desenvolvida.

O sistema segue os seguintes passos de funcionamento: configuração do sistema carregando as ontologias, inicialização e verificação de consistência das ontologias através do classificador DL, instanciação de sessões associadas a pedidos de acesso.

Os autores referem que seria interessante acrescentar ao modelo linguagens baseadas em regras, no entanto o tema precisa de mais discussão.

2.2 Triple-stores

Existem variados triple-stores, nesta tese/dissertação foram estudados o SwiftOWLIM, BigOWLIM e Virtuoso.

2.2.1 SwiftOWLIM

SwiftOWLIM é um triple-store com licença pública que disponibiliza a interface Sail API (ver secção 2.3.3). O repositório foi construído para um volume médio de dados (até 10 milhões de factos) e para fazer protótipos.

Tem como características:

- Motores de RDF nativos implementados em Java compiláveis com Sesame;
- Suporte robusto para as semânticas RDFS, OWL Horst and OWL 2 RL;
- Classificação e pesquisa são efectuadas em memória;
- Tem uma estratégia de persistência de forma a preservar os dados e a sua consistência;
- O carregamento dos dados e classificação é bastante rápido;
- Fácil configuração.

2.2.2 BigOWLIM

BigOWLIM[7] é um triple-store de licença comercial e implementando segundo a Sail API. O repositório foi construído para suportar grande volume de dados e pesquisas intensivas. Tem como características:

- Motores de RDF nativos implementados em Java compiláveis com Sesame;
- Suporte robusto para as semânticas RDFS, OWL Horst and OWL 2 RL;

Foi desenhado com um sistema de gestão de base de dados, o que torna possível as operações:

- Índices baseados em ficheiros, permitindo trabalhar com biliões de factos mesmo em máquinas pessoais;
- Índices especiais e técnicas de optimização de pesquisa garantem pesquisas rápidas para grandes volumes de dados;
- Optimização para owl: sameAs (identificador de igualdade) para aumentar a eficiência das tarefas de integração de dados;
- Validação eficiente de inferências inválidas, o que permite operações de eliminação mais eficientes.

2.2.3 Virtuoso

Virtuoso[8] é um quad-store de licença comercial que pode ser utilizador como motor das frameworks RDF Sesame e Jena. Tem as seguintes características:

- Suporta a linguagem de pesquisa SPARQL;
- Suporta o protocolo SPARQL;
- Permite integrar SPARQL com SQL;
- Permite indexação de Bitmaps e gestão de dados RDF;
- Implementação do projecto SIMILE HTTP-based Semantic Bank API;

[7] http://www.ontotext.com/owlim

[8] http://virtuoso.openlinksw.com/

- Implementação de HTTP baseado em RDF NET API;
- Método de inserção em RDF.
- Os dados RDF são acessíveis via ODBC, JDBC, ADO.NET OLE DB, e XMLA drivers.

2.2.4 Sumário e comparação

Na tabela seguinte faz-se um resumo e sistematização de várias características significativas dos triple-stores descritos anteriormente, aos quais se juntam outros três: Jena SDB, Jena TDB e Sesame.

Triple-store	**Interfaces**			**Inferência**[9]	**#Triplos**
	Nativa	**API**	**Query**		
Jena SBD	No	Jena API	Yes	Memória	
Jena TDB	Yes	Jena API	Yes	Memória	1,7 B
Sesame	Yes	Sesame	Yes	Memória	~70 M
Virtuoso	Yes	SAIL			+10 B
SwiftOWLIM	Yes	SAIL		Memória	+10 M
BigOWLIM	Yes	SAIL		Persistente	+10 B

2.3 Soluções (frameworks) para acesso a repositórios RDF

Existem variadas frameworks para acesso a repositórios RDF, i.e. triple-stores. Nesta tese/dissertação foram estudados o OWL API, Sesame e Jena.

2.3.1 OWL API

OWL API é um triple-store implementado em Java que trabalha com os factos em memória. Possui funcionalidades como:

- Interpretador e escritor de RDF/XML;
- Interpretador e escritor de OWL/XML;
- Interpretador e escritor da sintaxe funcional de OWL;
- Interpretador e escritor de Turtle;
- Interpretador de KRSS (???);
- Interpretador do formato OBO Flat;
- Interfaces para trabalhar com os classificadores FaCT++, HermiT, Pellet and Racer.

[9] Suporte físico dos factos no momento da execução do processo de inferência.

2.3.2 Jena

É um triple-store desenvolvido em java para a criação de aplicações Web semânticas. Possui as seguintes funcionalidades:

- Fornece APIs para trabalhar com RDF, RDFS and OWL, SPARQL e incluí um motor de inferência baseado em regras;
- RDF API;
- Leitura e escrita de RDF para RDF/XML, N3 e N-Triples;
- OWL API;
- Armazenamento em memória e repositório físico;

Motor de pesquisa para SPARQL.

2.3.3 Sesame

O Sesame[10] é um triple-store desenvolvido em Java que necessita a ligação a outros componentes (plug-in). A orientação do funcionamento depende da implementação do componente. Possui as seguintes funcionalidades:

- Tem duas interfaces de comunicação, i.e. Sail API e Repository API;
- Permite configurar e estender os mecanismos de armazenamento, inferência, formatos RDF, resultados das pesquisas e linguagens de pesquisa;
- API de pesquisa parecida com JDBC;
- APIs de sistema simples;
- Interface HTTP que suporta o protocolo SPARQL para RDF;
- Suporta pesquisa em SPARQL e SeRQL;
- Armazenamento em memória e em repositório físico;
- Motores de inferência para esquemas RDF;
- Suporta formatos de pesquisa mais populares.

A escolha da framework Sesame deve-se ao facto de ser possível escolher qual a implementação de acesso a repositórios RDF que se pretende utilizar. Foi escolhida a implementação SwiftOWLIM pois, das opções disponíveis, era a única com licença pública. As outras opções de implementação avaliadas foram o Virtuoso e o BigOWLIM.

[10] http://www.openrdf.org/doc/sesame2/api/

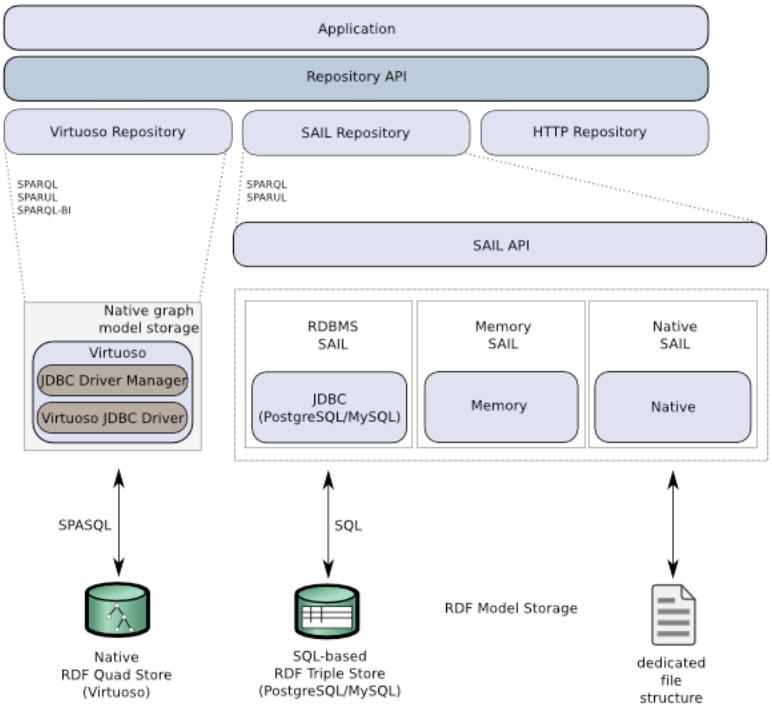


Figura 22 - Arquitectura conceptual de triple-store disponibilizando Sail API (fonte [11])

A figura anterior demonstra como é possível efectuar a integração com o Sesame. É necessário ter uma implementação de Repository API, nesta imagem são demonstradas três possíveis implementações Virtuoso Repository, Sail Repository e HTTP Repository. Este trabalho incide sobre a Sail API, logo é a implementação de Repository API que é importante mais concretamente a Sail API. A Sail API é implementada por vários repositórios semânticos de forma a permitir o acesso aos factos (e.g. SwiftOWLIM e BigOWLIM). São estas implementações que depois definem por exemplo o tipo de armazenamento (memoria ou repositório físico).

[11] http://docs.openlinksw.com/virtuoso/rdfnativestorageproviders.html

Capítulo 3

Requisitos

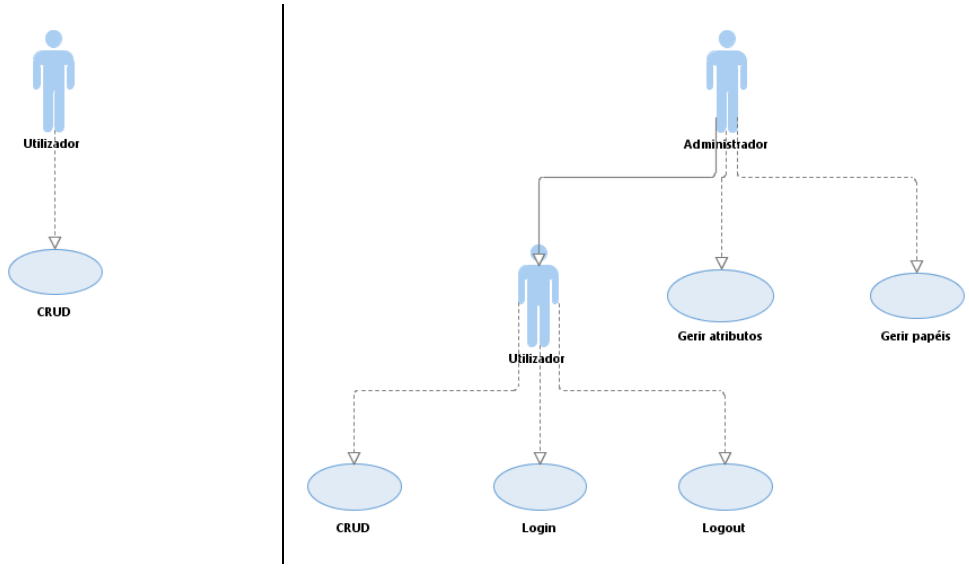
Neste capítulo pretende-se sistematizar os requisitos a considerar, em nomenclatura de engenharia de software. Na primeira sub-secção são apresentados os requisitos funcionais e na segunda são apresentados os requisitos não funcionais.

3.1 Requisitos funcionais

A grande maioria dos triple-stores e quad-stores não incluem funcionalidades que permitam efectuar o controlo de acesso aos factos do repositório RDF. É necessário fazer evoluir o dito triple-store de forma a incluir novas funcionalidades de:

- Autenticação
- Autorização, i.e. acesso controlado a factos no repositório
- Gestão de papéis
- Gestão de atributos

A Figura 23 representa o diagrama UML de casos de uso existente, com o requerido.



a) b)

Figura 23 - Interacção com o Sesame nativo (a) e utilizando OntologyAccessModel (b)

Como demonstra a figura anterior, a interface Sail permite manipular e consultar factos num repositório. Torna-se necessário acrescentar novas funcionalidades às existentes, i.e. login, logout, gestão de papéis e acesso controlado a factos. É a partir das funcionalidades de "Login" e "Logout" que o sistema consegue identificar qual o utilizador autenticado, e a partir dessas efectuar o controlo de acessos. A funcionalidade "Gerir papéis" é relevante pois a configuração de um papel é um processo algo complexo, onde a complexidade varia consoante os modelos de papéis. Decidiu-se então que o sistema deve fornecer um mecanismo de gestão de papéis ao utilizador, de forma a tornar o sistema mais amigável e consequentemente aumento as suas hipóteses de adopção. A gestão de atributos é semelhante, mas neste casos relativamente a atributos (de sujeitos e de recursos/factos).

3.2 Requisitos não funcionais

Os triple-stores e quad-stores são os mecanismos de base de armazenamento da Web Semântica e de sistemas empresariais semânticos, no qual existe a necessidade de controlar o acesso a factos.

Os factos não estão limitados à forma de representação, i.e. os factos podem ser representados por triplos, quádruplos, quíntuplos, etc. que o modelo tem de responder sempre à totalidade do objectivo.

Os esquemas de dados devem estar representados na forma de ontologias com a expressividade mínima ALC (Baader et al. 2003).

O produto tem de permitir utilizar qualquer algoritmo e abordagem de controlo de acessos, i.e. tem de suportar abordagens de controlo de acesso orientadas a atributos, papéis, utilizador, etc. e tem de permitir utilizar qualquer algoritmo de controlo de acessos.

O modelo proposto deve ser passível de ser incorporado em qualquer aplicação que seja desenvolvida respeitando a API Sail (e.g. SwiftOWLIM, Sesame, Virtuoso).

Qualquer modelo de papéis deve ser passível de ser incorporado no modelo proposto nesta tese/dissertação, i.e. os modelos de papéis são variáveis e de complexidades diferentes. Deve ser fornecido um modelo de papéis base que seja passível de ser estendido para qualquer outro modelo.

Capítulo 4

Design

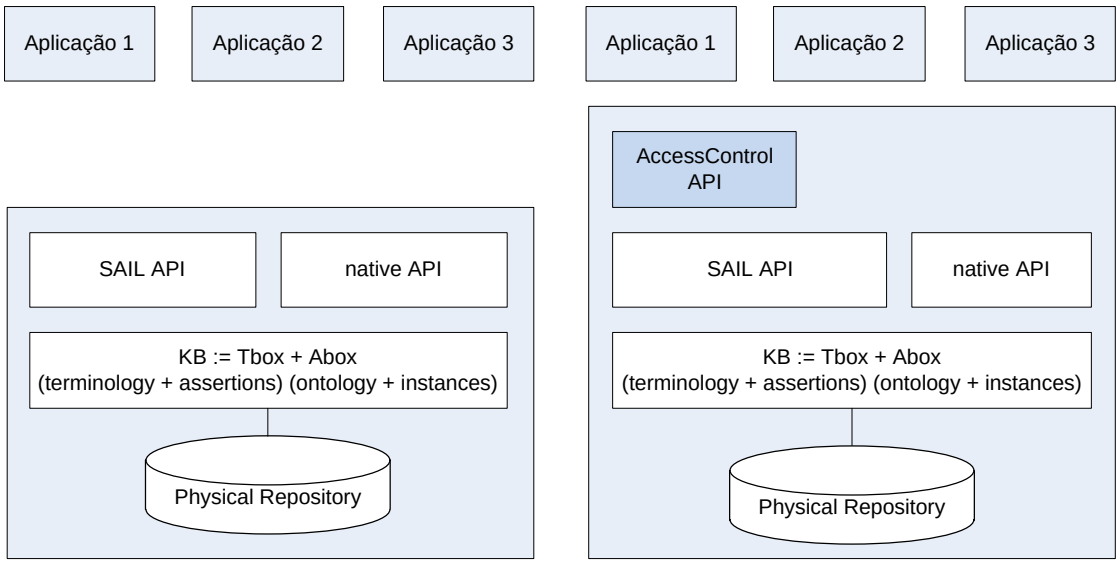
Neste capítulo descreve-se o design de software proposto para responder aos requisitos identificados e descritos na secção anterior.

4.1 Perspectiva geral

A solução preconizada deverá fornecer um mecanismo de autenticação do sujeito/utilizador e os seus papéis. Uma vez autenticado, a autorização de acesso a recursos está dependente dos privilégios atribuídos e da avaliação que o mecanismo de autorização fizer destes.

A evolução do sistema foi efectuada através da criação do componente Access Control API. Esta API caracteriza-se por:

- implementar a especificação JAAS no processo de autenticação;
- estender a Sail API em funcionalidades que garantam o acesso controlado a factos.



a) original b) proposto

Figura 24 – Representação de infra-estrutura de Access Control e comparação

O JAAS[12] (Java Authentication and Authorization Service) é uma especificação para serviços de controlo de autenticação e autorização, para ser usado por aplicações desenvolvidas em Java. Caracteriza-se por um fraco acoplamento entre a aplicação Java e o serviço de autenticação e autorização. Este modelo será o ponto inicial da solução preconizada.

Embora o serviço JAAS compreenda autenticação e autorização, apenas a autenticação será aplicada, uma vez que em JAAS a autorização diz respeito à autorização de execução de código e não propriamente à autorização de acesso a recursos, nomeadamente dados.

Assim sendo, enquanto o processo de autenticação obedecerá à especificação JAAS, o processo de autorização terá que adoptar outra abordagem. Recorrendo à nomenclatura de componentes do UML, a Figura 25 representa a perspectiva geral proposta.

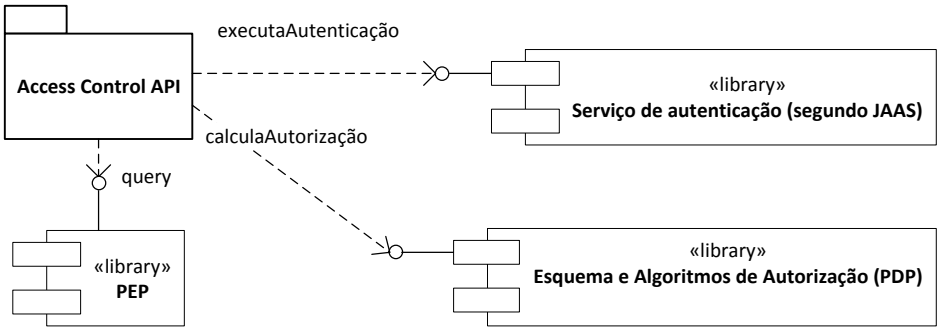


Figura 25 - Perspectiva geral de design

[12] http://download.oracle.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html

A proposta inclui portanto a actuação sobre três componentes:

- Triple-store, que deverá:
  - Solicitar autenticação antes de qualquer acesso;
  - Requerer ao PDP as permissões de acesso a recursos;
  - Disponibilizar um PEP que baseado na decisão do PDP, autoriza ou recusa o acesso a recursos;
- Serviço compatível com JAAS, que garante um mecanismo normalizado de autenticação de utilizadores e de acesso aos seus papéis (RBAC) e/ou atributos (ABAC);
- Módulo de autorização, que inclui os modelos de dados e os algoritmos de avaliação de permissões.

Nas secções seguintes descreve-se com mais detalhe a solução preconizada recorrendo a diagramas de classes e de actividade.

4.2 Conexão usando autenticada

Autenticação é o acto da identificação de quem está ou pretende utilizar o sistema.
Sail API é um contracto para manipular RDF proposto no âmbito do Sesame, e que é disponibilizado por muitos triple-stores como forma de aceder ao repositório. É esta interface que é disponibilizada às aplicações para acesso ao repositório. A Figura 26 representa em UML a interface.

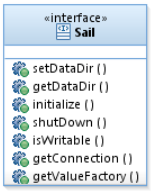


Figura 26 – Interface Sail

O processo de instanciação de Sail é representado no diagrama de sequência da Figura 27.

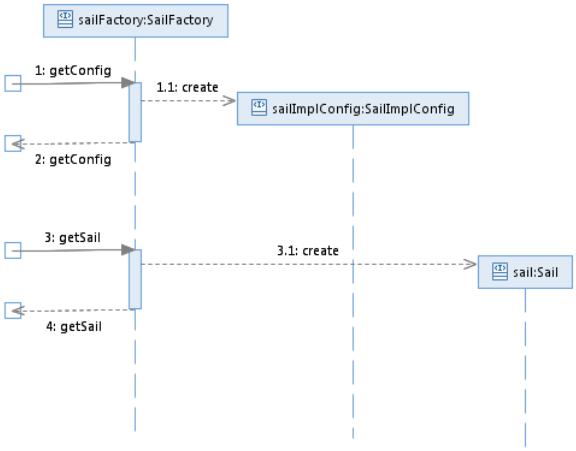


Figura 27- Instanciação (conceptual) de Sail

Como o nome indica o contrato SailFactory é uma fábrica (factory) (Gamma, Helm, Johnson & Vlissides 1994a) para a instanciação da interface Sail. As implementações da fábrica SailFactory têm a responsabilidade de instanciar implementações de SailImplConfig e Sail, além de identificar a que repositório pertencem. A interface SailImplConfig é responsável por validar se a configuração está coerente e disponibilizar a configuração de ligação ao repositório. A Figura 28 representa em notação UML o diagrama de classes aplicadas no diagrama de sequência anterior (Figura 27).

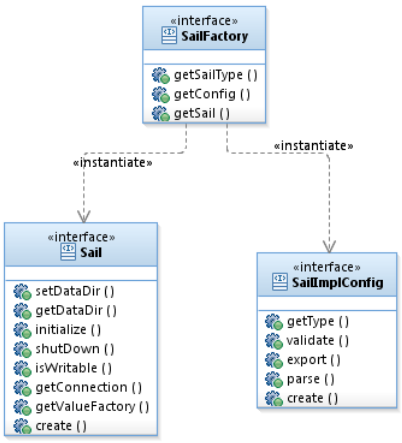


Figura 28 - Diagrama de classes de instanciação (conceptual) de Sail

No entanto, este processo e (por conseguinte) a interface Sail, não são suficientes para atingir os objectivos propostos de autenticação e acesso condicionado, uma vez que não requerem em altura alguma a autenticação do sujeito. Para responder aos requisitos, sugere-se a evolução deste processo pela inclusão do serviço de autenticação compatível com a especificação JAAS.

O design terá de evoluir de forma que a interface de acesso aos factos garanta acesso condicionado segundos os privilégios do sujeito. Essa responsabilidade não está atribuída a nenhuma interface ou classe de Sail. As responsabilidades de acesso incondicional aos factos estão atribuídos a SailConnection, que por sua vez está dependente da interface Sail.

Consequentemente, o design terá de actuar nas responsabilidades de SailConnection, mas também nas responsabilidades de Sail.

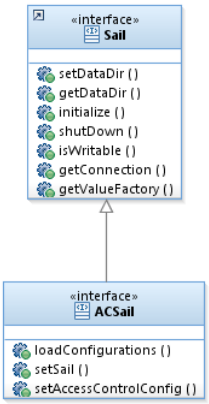


Figura 29 - Diagrama de classes: ACSail, Sail, ACSailImpl e suas relações

O processo de instanciação de ACSail requer parâmetros que não eram exigidos para Sail, pelo que se propõe o processo seguinte (Figura 30).

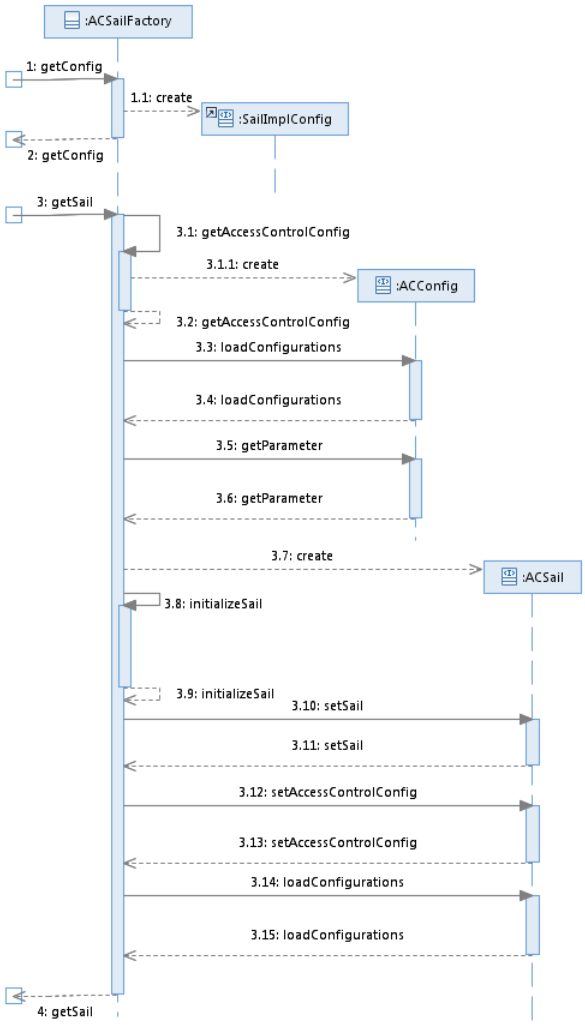


Figura 30 – Processo de instanciação de ACSail

O processo foi modificado para que a fábrica SailFactory seja implementada pela fábrica ACSailFactory que tem a responsabilidade de instanciar ACSail. As implementações de ACSail são compostas por uma instância de Sail do repositório a que se pretender condicionar os acessos (e.g. SailImpl do SwiftOWLIM). Sendo que a fábrica ACSailFactory é uma classe abstracta que tem de ser estendida de forma a fornecer:

- Uma implementação de ACConfig;
- Uma implementação de SailImplConfig do triple-store;
- Uma implementação de Sail do triple-store.

A forma de fornecer as implementações de SailImplConfig, Sail e ACConfig é através de uma classe que estenda ACSailFactory e faça a implementação dos métodos getConfig(), initializeSail(SailImplConfig SailImplConfig) e getAccessControlConfig(). Os dois primeiros métodos devem delegar na fábrica do triple-store a responsabilidade de instanciar as interfaces SailImplConfig e Sail respectivamente. Estes métodos são portanto dependentes do repositório em que a Access Control API vier a ser adoptada.
Assim, para além das responsabilidades herdadas de SailFactory, ACSailFactory tem ainda as seguintes responsabilidades:

- Instanciar SailImplConfig que será usada para posteriormente instanciar Sail do repositório (passo 1.1 do diagrama da Figura 30);
- Carrega as configurações de execução do controlo de acesso usando para isso a interface ACConfig (passos 3.1 a 3.5);
- Instanciar ACSail (3.6), sua configuração com instância de Sail (3.9) e a sua configuração (3.11 a 3.14).

ACConfig é portanto responsável por carregar as configurações de execução do controlo de acesso, algo que é dependente de repositório. As configurações a carregar incluem: nomes qualificados das implementações das interfaces do modelo, configuração dos papéis e/ou atributos.
Note-se que ACSail é utilizada para o acesso a qualquer repositório que respeite a API Sail. Ou seja, está completamente desacoplada das implementações dos repositórios.
A Figura 31 representa em notação UML o diagrama de classes aplicadas no diagrama de sequência anterior (Figura 30).

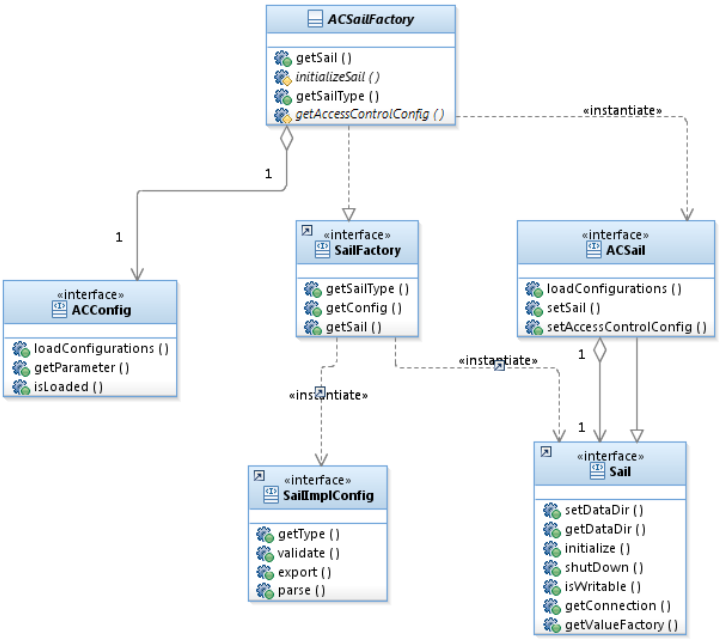


Figura 31 – Diagrama de classes usada no processo de instanciação de ACSail

A interface Sail permite o acesso à base de conhecimento e em particular aos factos aí existentes através duma conexão (interface SailConnection). A importância de SailConnection advém de disponibilizar métodos de acesso ao repositório, nomeadamente métodos de inquérito (e.g. evaluate()). A interface SailConnection é representada na Figura 32.

Figura 32 - Interface SailConnection

Estando esta interface dependente da configuração efectuada por Sail, não permite identificar qual o utilizador autenticado e respectivos papéis, bem como por consequência condicionar o acesso aos factos, mediante os privilégios do sujeito (Figura 33).

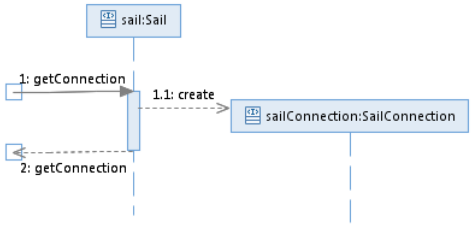


Figura 33 - Instanciação (conceptual) de SailConnection

Foi necessário evoluir o design para que ACSail consiga obter uma conexão que garanta acesso controlado ao repositório. Para isso definiu-se a interface ACSailConnection que garanta o acesso controlado a factos de acordo com o utilizador e respectivos privilégios. O processo proposto está representado no diagrama de sequência da Figura 34.

Introduz-se aqui a classe ACSailImpl que é (uma que classe que é) uma implementação da interface ACSail, notavelmente independente de repositório. ACSailImpl funciona como proxy de uma instância de Sail.

A principal implementação de ACSailImpl faz de ACSail é o método getConnection(). Este método tem a responsabilidade de instanciar ACSailConnection, sendo que para o efeito necessita de obter o sujeito (utilizador) autenticado, os papéis e os atributos que lhe estão associados e fornecê-los a ACSailConnection.
Para isso recorre à interface ACAuthentication que define métodos de acesso à informação sobre o utilizador autenticado (getSubject), respectivos papéis (getRolesNames) e respectivos atributos (getAttributes). Além disso, ACAuthentication é o ponto de ligação com a especificação JAAS. A interface ACAuthentication desempenha portanto o papel de PIP.

Neste processo a classe ACSailImpl é ainda responsável por fornecer à instância de ACSailConnection uma ligação válida ao repositório semântico (SailConnection) e a fábrica de objectos que correspondem a factos do repositório semântico. De notar também que neste processo a classe SailConnectionImpl é substituída pela classe ACSailConnectionImpl que implementa a interface ACSailConnection que por sua vez estende a interface SailConnection.
ACSailConnectionImpl é independente de repositório.

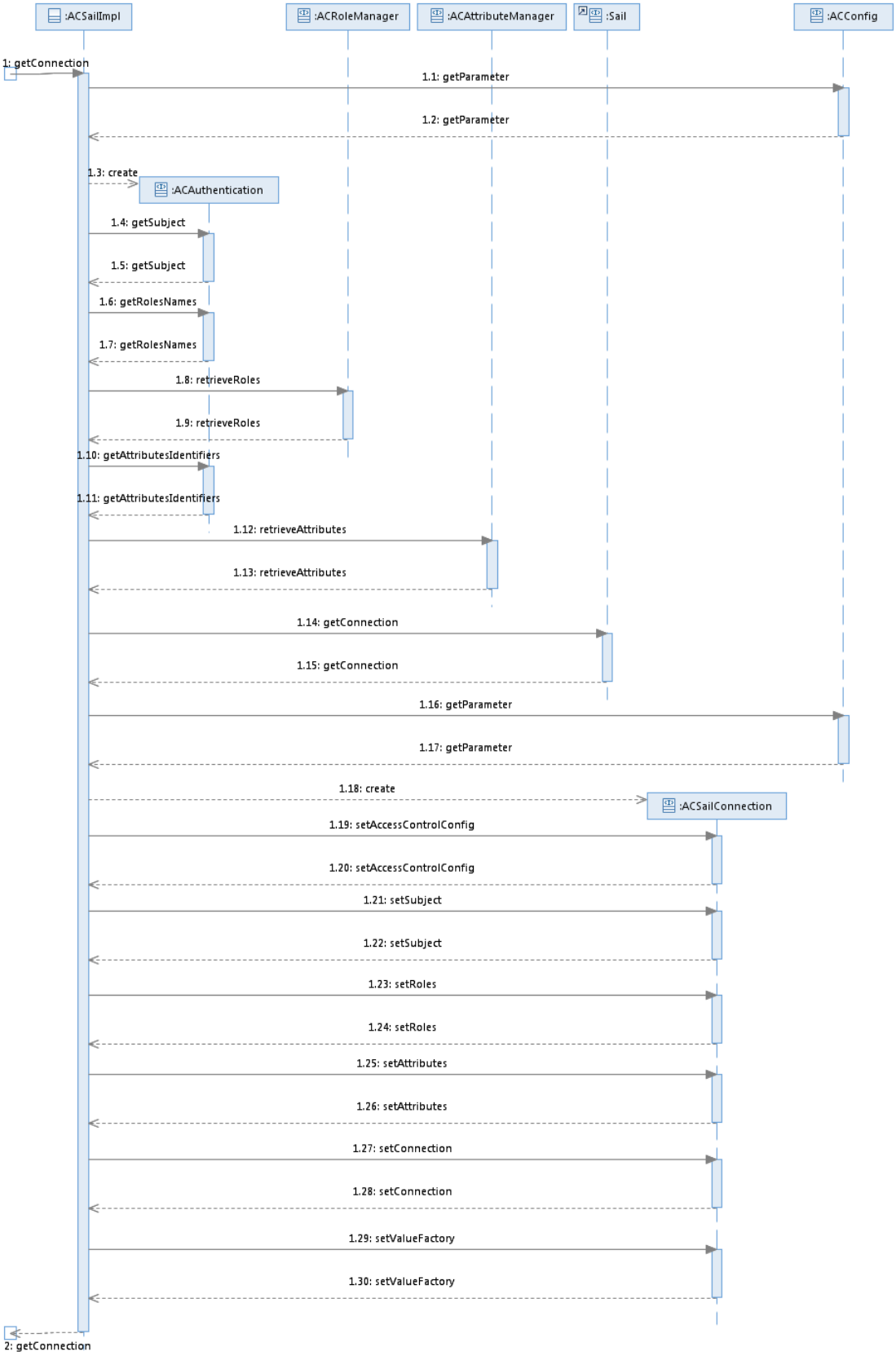


Figura 34 - Obter conexão do ACSailConnection

A Figura 35 representa o diagrama UML das classes aplicadas no diagrama de sequência anterior (Figura 34).

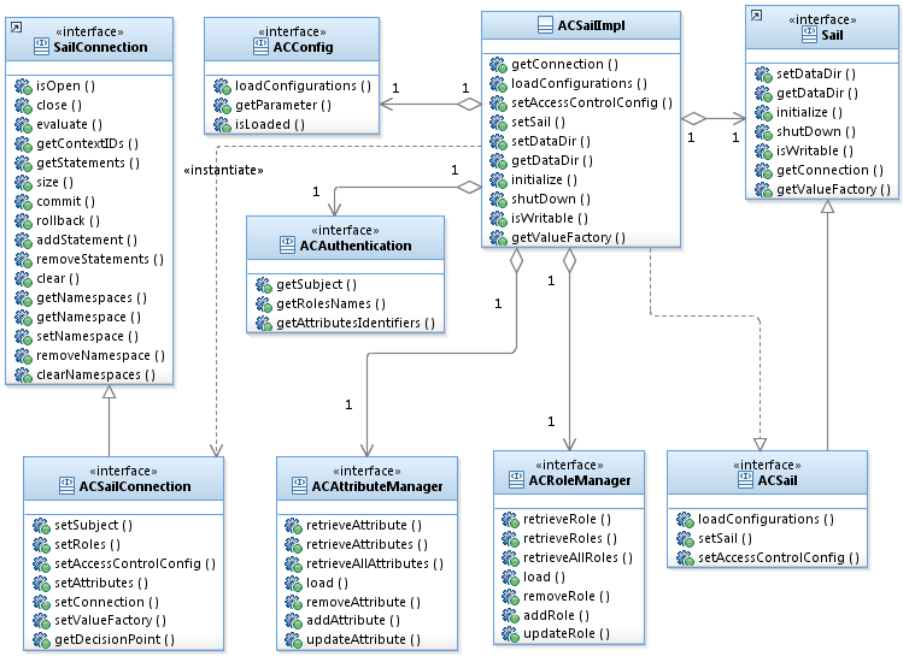


Figura 35 – Diagrama de classes para obter conexão do ACSailConnection

Note-se que este processo de instanciação de ACSailConnection pode ser executado em diferentes momentos, e não necessariamente imediatamente após a autenticação.

4.3 CRUD: Create, Retrieve, Update and Delete

Nesta secção descreve-se o design proposto para responder ao caso de uso CRUD (Grand 2001), considerando o design já descrito.

Qualquer que seja o comando a executar (i.e. Create, Read, Update ou Delete), a execução desse e o respectivo resultado devem obedecer aos privilégios do sujeito.

É a interface SailConnection que define a responsabilidade de execução dos comandos CRUD, nomeadamente através do comando evaluate(). Esta interface não pressupõe a identificação do utilizador nem dos respectivos papéis, não garantindo consequentemente o controlo de acesso. Permite apenas o acesso incontrolado ao repositório (Figura 36).

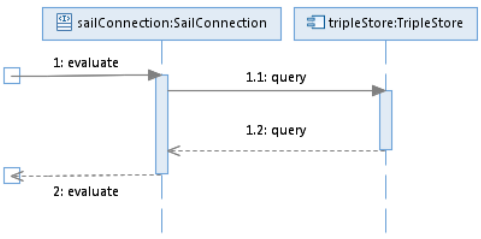


Figura 36 - Efectuar pesquisa do Sail API

O design proposto foca-se portanto nas alterações ao nível da conexão, substituindo a interface SailConnection por ACSailConnection. Para além das responsabilidades de SailConnection, ACSailConnection tem a responsabilidade de:

- Controlar a execução de comandos sobre o repositório;
- Controlar o acesso aos factos devolvidos.

Contudo, propõe-se a implementação independente de repositório do método evaluate() na classe ACSailConnectionImpl, que implementa a interface ACSailConnection. A Figura 37 é o diagrama de sequência implementado do método evaluate().

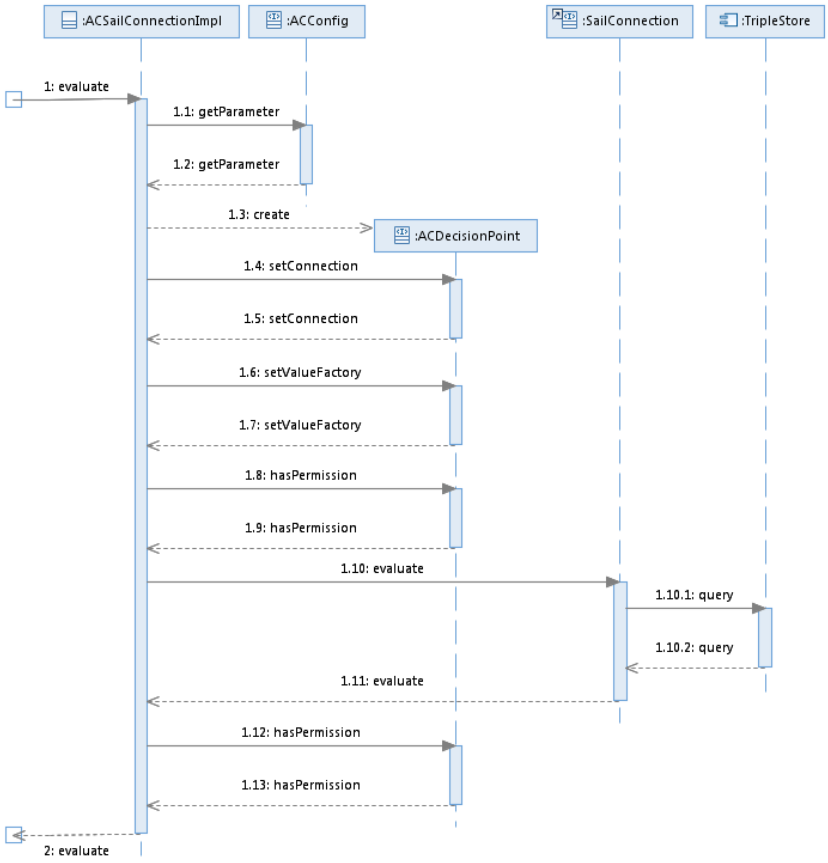


Figura 37 - Efectuar pesquisa do ACSailConnectionImpl

Este método caracteriza-se por delegar numa instância da interface ACDecisionPoint a responsabilidade da decisão sobre execução do comando e sobre o acesso aos recursos. Para isso:

- Instancia a interface ACDecisionPoint (passo 1.3 da Figura 37);
- Fornece à implementação da interface ACDecisionPoint a instância de SailConnection (passo 1.4), para que a instância tenha acesso (incontrolado) à informação do repositório, necessária para a decisão;

- Avalia a permissão de execução do comando (passo 1.8), delegando em ACDecisionPoint essa responsabilidade;
- Executa o comando via a instância de SailConnection do repositório (passo 1.10);
- Avalia as permissões de acesso aos factos resultantes da execução do comando (passo 1.12), delegando em ACDecisionPoint essa responsabilidade.

Estas responsabilidades fazem a interface ACSailConnection desempenhar o papel de PEP, dado que a implementação desta interface é responsável por aplicar a tomada de decisão.
A interface ACDecisionPoint representa o modelo e algoritmo de controlo de acessos, tendo como responsabilidades:

- Determinar as permissões de execução do comando pelo sujeito;
- Determinar as permissões de acesso a factos pelo sujeito.

Estas responsabilidades fazem da interface ACDecisionPoint desempenhar o papel de PDP, dado que a implementação desta interface é responsável pela tomada de decisão.
A Figura 38 representa o diagrama UML das classes aplicadas no diagrama de sequência anterior (Figura 37).

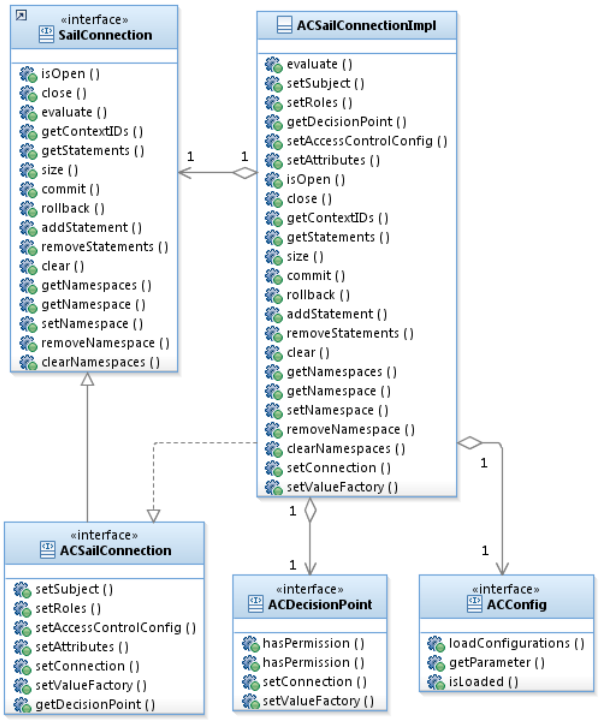


Figura 38 – Diagrama de classes de pesquisa de ACSailConnectiomImpl

4.4 Gestão de papéis

O modelo de papéis é a estrutura em que assenta a definição de configuração do papel, i.e. se é possível existir hierarquias entre papéis, restrições entre papéis, etc. Sabendo que a definição

base de papel tem de ser respeitada (nome e conjuntos de permissões). Para gerir os factos necessários a tais processos, sugere-se a adopção duma interface que todas as aplicações sigam.

Para isso foi definida a interface ACRoleManager que segue o padrão CRUD, i.e. tem a responsabilidade de criar addRole(), disponibilizar (retrieveRoles()), actualizar (updateRole())e apagar (removeRole()) papéis do sistema.

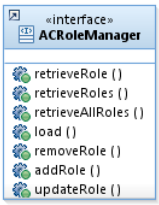


Figura 39 – Interface ACRoleManager

Note-se que esta interface não define a estrutura ou interface do conceito de papel (role), pois essa dimensão é tipicamente dependente do modelo e algoritmos de controlo de acesso.Gestão de atributos

O modelo de atributos é a estrutura em que assenta a definição de controlo de acessos por atributos. Tal como para papéis, foi definida e sugere-se a adopção duma interface que respeita o padrão CRUD, a interface ACAttributeManager.

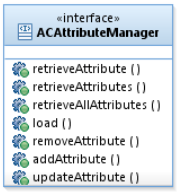


Figura 40 - Interface ACAttributeManager

A interface ACAttributeManager define a responsabilidade de adicionar (addAttribute()), disponibilizar (retrieveAttribute()), actualizar (updateAttribute()) e apagar (removeAttribute()) os atributos de acordo com o modelo definido.

Neste projecto não é introduzido qualquer estrutura ou interface de atributos, pois é tipicamente dependente do modelo e algoritmo de controlo de acessos.

4.5 Classes por serviços de controlo de acesso

É relevante classificar cada componente no que respeita aos serviços de controlo de acesso que disponibiliza, dado que identifica quais os objectos que são utilizados para executar e configurar o controlo de acesso (Figura 41).

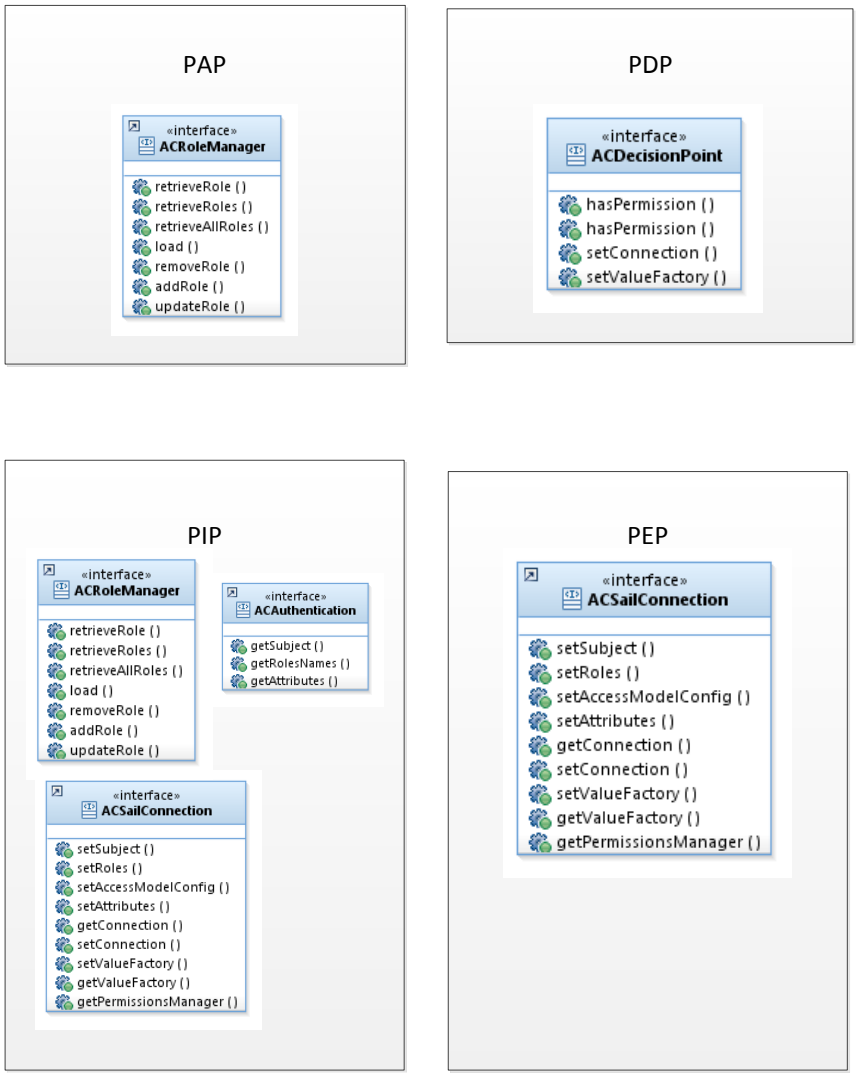


Figura 41 - Serviços de controlo de acesso

A interface ACRoleManager e a classe ACRoleUIManager permitem gerir todas as regras de acesso de um papel aos recursos, o que os fazem desempenhar o papel PAP.

A interface ACPermissionsDecider tem responsabilidade de verificar se um dado utilizador tem acesso, pelo que faz dela um PDP.

A interface ACSailConnection tem como uma das responsabilidades aplicar a tomada de decisão do PDP, portanto desempenha o papel de PEP.

As interfaces ACRoleManager, ACSailConnection e ACAuthentication fornecem informação ao PDP para que possa efectuar a tomada de decisão fazendo que estas interfaces desempenhem o papel de PIP.

Capítulo 5

Implementação

O design descrito no capítulo anterior é um modelo e interface programática genérica, baseada no facto do triple-store disponibilizar uma interface Sail. Não se trata portanto duma solução para qualquer triple-store específico. Neste capítulo descreve-se a implementação do design anterior, para o triple-store SwiftOWLIM.

Pretende-se integrar o SwiftOWLIM num ambiente J2EE e com autenticação, neste caso WebSphere Application Server (WAS). O modelo e algoritmo de controlo de acessos adoptado é baseado naqueles descrito na secção 2.1.3, portanto um algoritmo RBAC. O modelo e algoritmo de controlo de acesso representa o modelo de permissões que dá origem ao algoritmo de controlo de acesso. A gestão de atributos vai ser ignorada neste capítulo de forma a não criar entropias.

Os diagramas de classes e sequência apresentados neste capítulo são orientados às implementações dos contractos de Sail API e do modelo de controlo de acessos.

O SwifOWLim usa como implementação de Sail a classe SailImpl. A Figura 42 representa em UML a interface e a sua implementação.

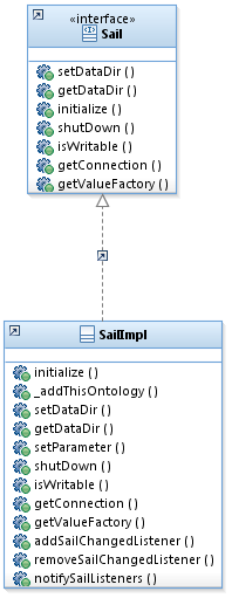


Figura 42 - Implementação de SailImpl do SwiftOWLIM

O processo de instanciação de SailImpl é representado no diagrama de sequência da Figura 43.

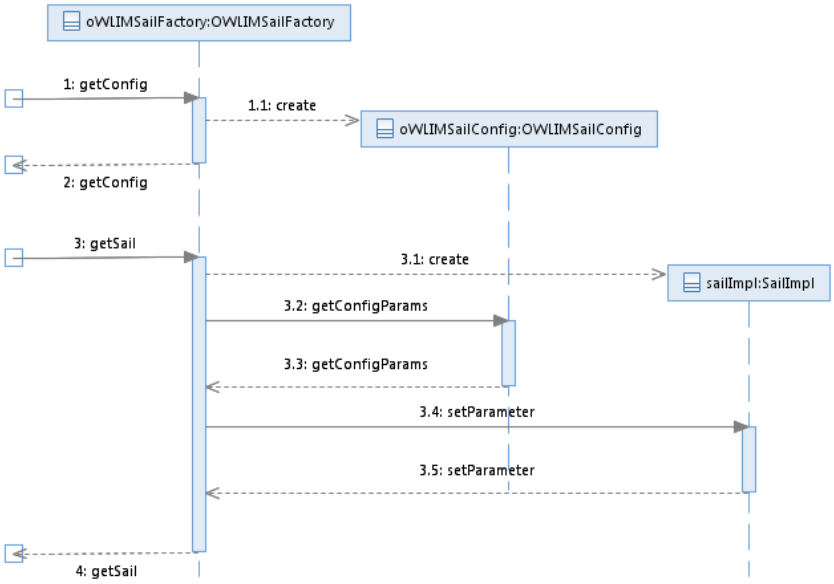


Figura 43 - Instanciação de SailImpl em SwiftOWLIM

Este processo é a implementação em SwiftOWLIM do processo descrito no diagrama apresentado na Figura 27. As interfaces de Sail API foram substituídas por implementações concretas orientadas ao triple-store. Ou seja:

- SailFactory foi substituída por OWLIMSailFactory;
- SailImplConfig foi substituída por OWLIMSailConfig;
- Sail foi substituída por SailImpl.

A Figura 44 representa em notação UML o diagrama de classes aplicadas no diagrama de sequência anterior (Figura 43).

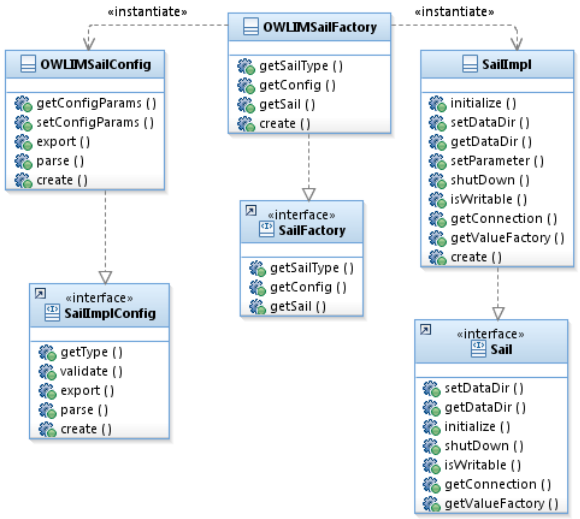


Figura 44 – Diagrama de classes de implementação da Sail API no SwiftOWLIM

Como referido na secção anterior, este modelo não permite controlar os acessos ao repositório. Para permitir condicionar os acessos ao triple-store é necessário transformar o processo anterior (Figura 43) no processo demonstrado na Figura 30. Para isso utiliza-se uma classe que estende ACSailFactory direccionada para o SwiftOWLIM (ACSailFactoryImpl). Essa fábrica tem a responsabilidade de instanciar correctamente OWLIMSailConfig, ACSailImpl e ACConfigImpl, como demonstra o diagrama de sequência da Figura 45.

Neste processo ACSailFactoryImpl delega na fábrica OWLIMSailFactory a responsabilidade de instanciação de uma implementação de SailImplConfig, sendo ela OWLIMSailConfig. A instanciação de ACConfigImpl é efectuada através da implementação do método getAccessControlConfig(). No restante processo existe o acréscimo de comportamento relativo à instanciação de Sail. Isto é, ACSailFactoryImpl também delega na fábrica OWLIMSailFactory a responsabilidade de obter uma implementação de Sail do triple-store.

Por fim, como já foi mencionado, ACSail é implementada por ACSailImpl que ao carregar as suas configurações irá também carregar as configurações de ACRoleManagerImpl. ACRoleManagerImpl é a implementação de ACRoleManager que contém o modelo de papéis utilizado para esta demonstração. Neste processo ACSailImpl não carrega as configurações de ACAttributeManager dado que se pretende trabalhar apenas com papéis.

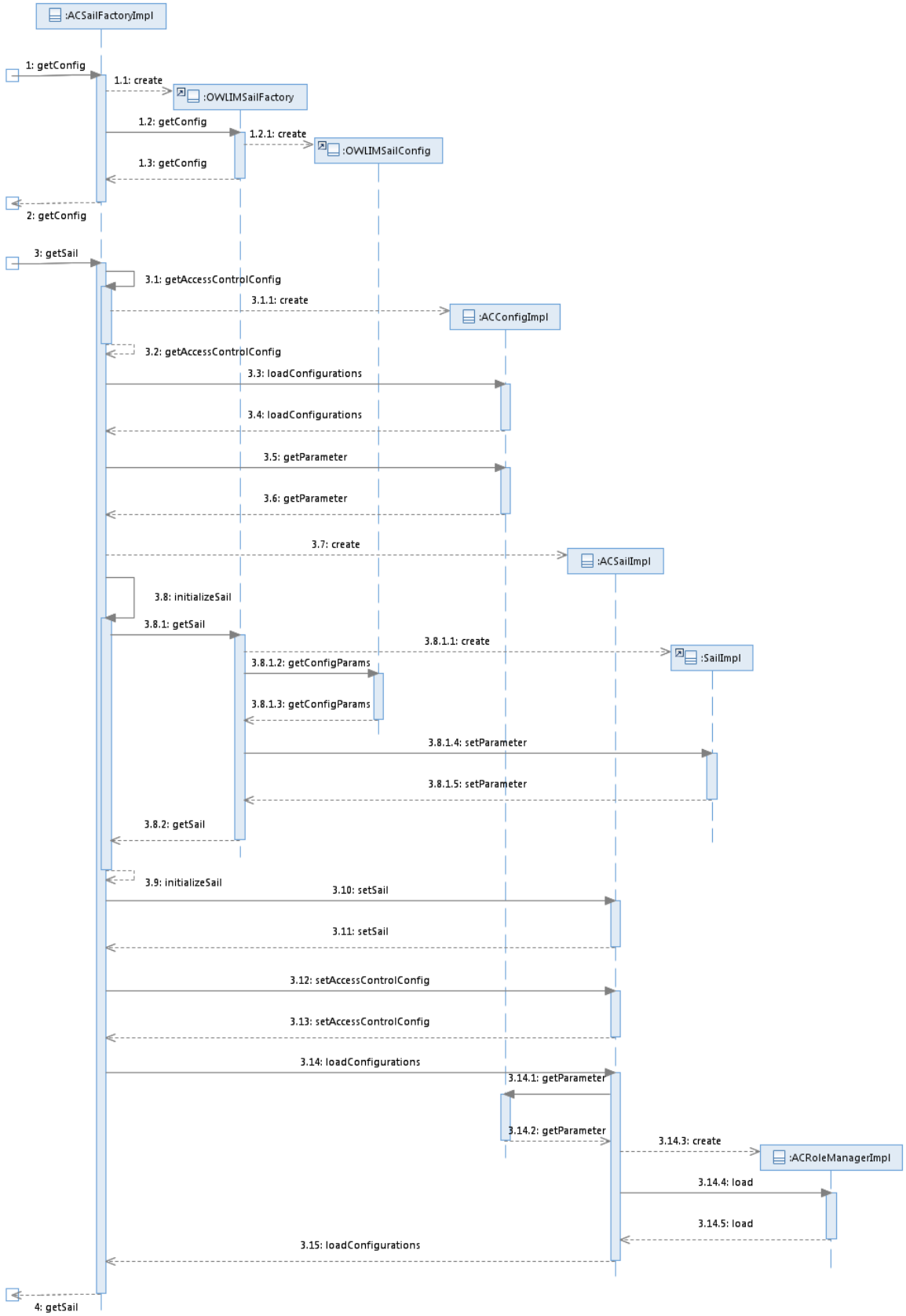


Figura 45 - Instanciação de ACSailImpl em SwiftOWLIM

A Figura 46 representa em notação UML o diagrama de classes aplicadas no diagrama de sequência anterior (Figura 45).

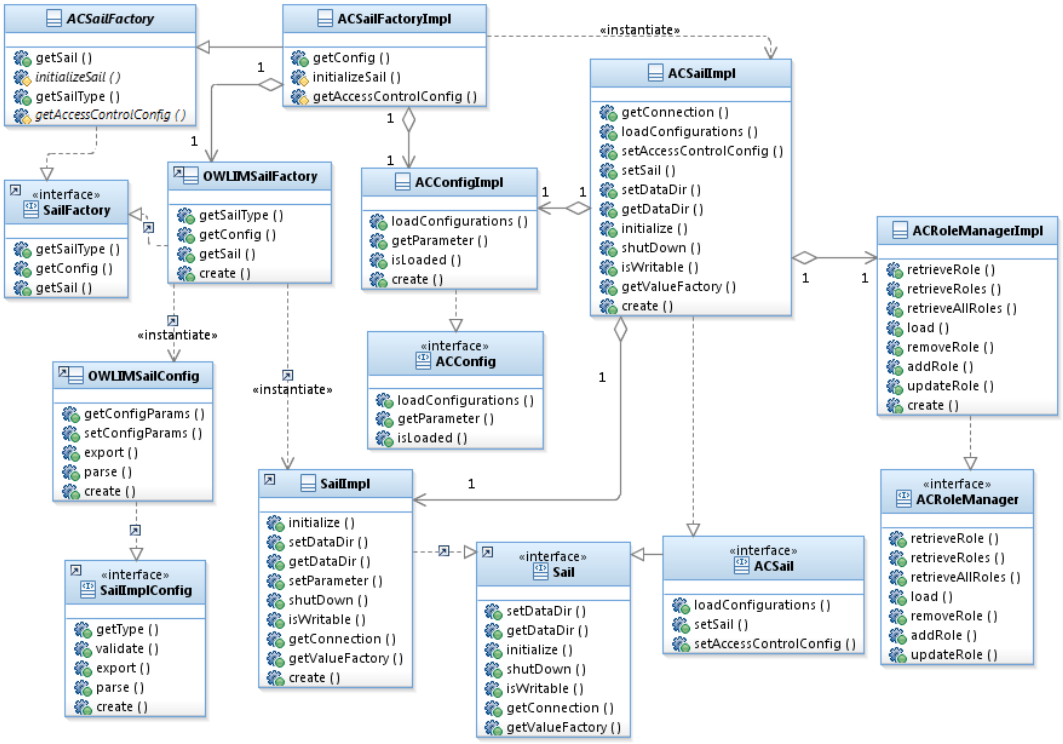


Figura 46 - Diagrama de classes de instanciação de ACSailImpl para SwiftOWLIM

Para que seja possível aceder ao repositório é necessário que seja disponibilizada uma conexão. O SwiftOWLIM disponibiliza a SailConnectionImpl que implementa SailConnection, que como já foi demonstrado, não permite autenticação nem autorização. A Figura 47 é o diagrama de sequência que representa o processo de instanciação de SailConnectionImpl.

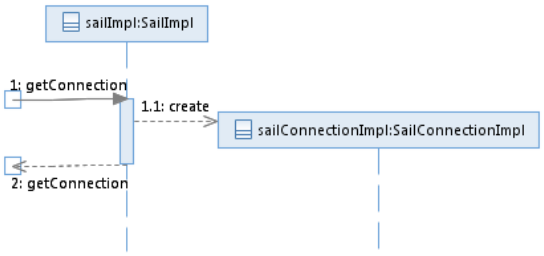


Figura 47 – Processo de instanciação de SailConnectionImpl

No processo base de Sail API (Figura 33), a interface Sail tem como responsabilidade disponibilizar uma conexão (i.e. SailConnection) para o triple-store SwiftOWLIM. No SwiftOWLIM essa conexão é uma instância de SailConnectionImpl, que é responsável pelo acesso ao repositório.

A Figura 48 representa o diagrama de classes do diagrama de sequência anterior (Figura 48).

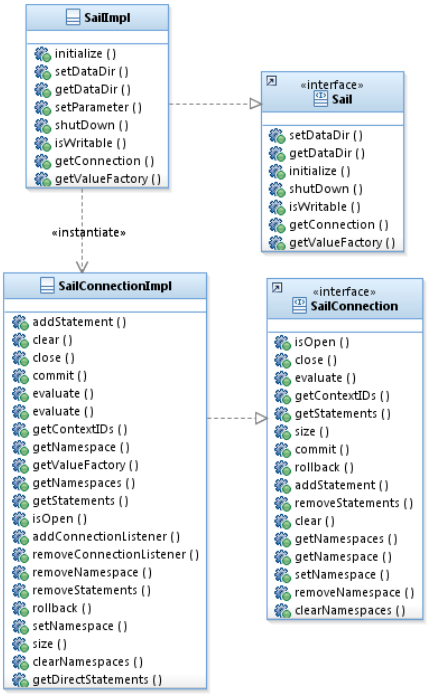


Figura 48 – Diagrama de classe usadas na instanciação de SailConnectionImpl

Para permitir condicionar o acesso ao repositório é necessário evoluir o processo acabado de apresentar (Figura 47) no processo demonstrado na Figura 34Para isto vai-se utilizar uma instância de ACSailConnectionImpl (que implementa ACSailConnection) uma vez que esta classe é independente do triple-store.

O próximo diagrama de sequência demonstra o processo para obter uma conexão de acesso controlado ao repositório. Uma vez que neste processo não existe qualquer referência ao modelo de atributos pois não se pretende trabalhar com atributos.

Por comparação com o diagrama de design da Figura 34, e exceptuando a não adopção de atributos, as alterações a este diagrama de implementação limitam-se à substituição das interfaces para as suas implementações, e respectivos mecanismos de instanciação.

No seguimento do processo de instanciação de ACSailImpl é agora utilizada essa instância de forma a obter uma conexão ao repositório. Isto é, pede à instância de SailImpl que lhe devolva uma conexão de acesso (incondicionado) ao repositório (SailConnectionImpl). Essa instância de SailConnectionImpl é então atribuída a ACSailConnectionImpl.

ACWASAuthentication é uma implementação de ACAuthentication orientada para o WebSphere Application Server (WAS), que permite a integração do serviço de autenticação com o do serviço WAS, que respeitam ambos a especificação JAAS. O SwiftOWLIM pode assim beneficiar da partilha do processo de autenticação com o WAS.

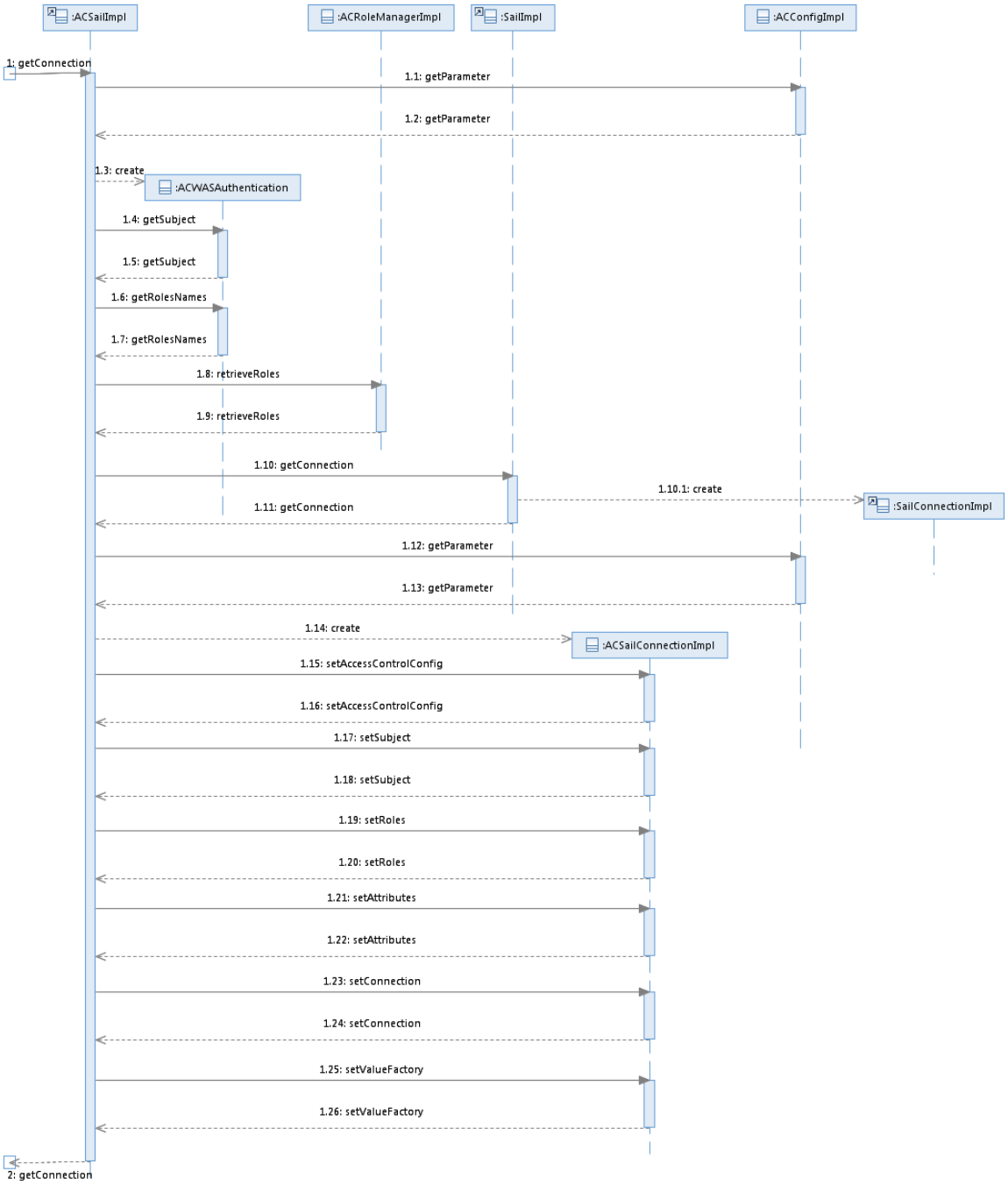


Figura 49 - Instanciação de ACSailConnectionImpl

A Figura 50 representa em notação UML o diagrama de classes aplicadas no diagrama de sequência anterior (Figura 49).

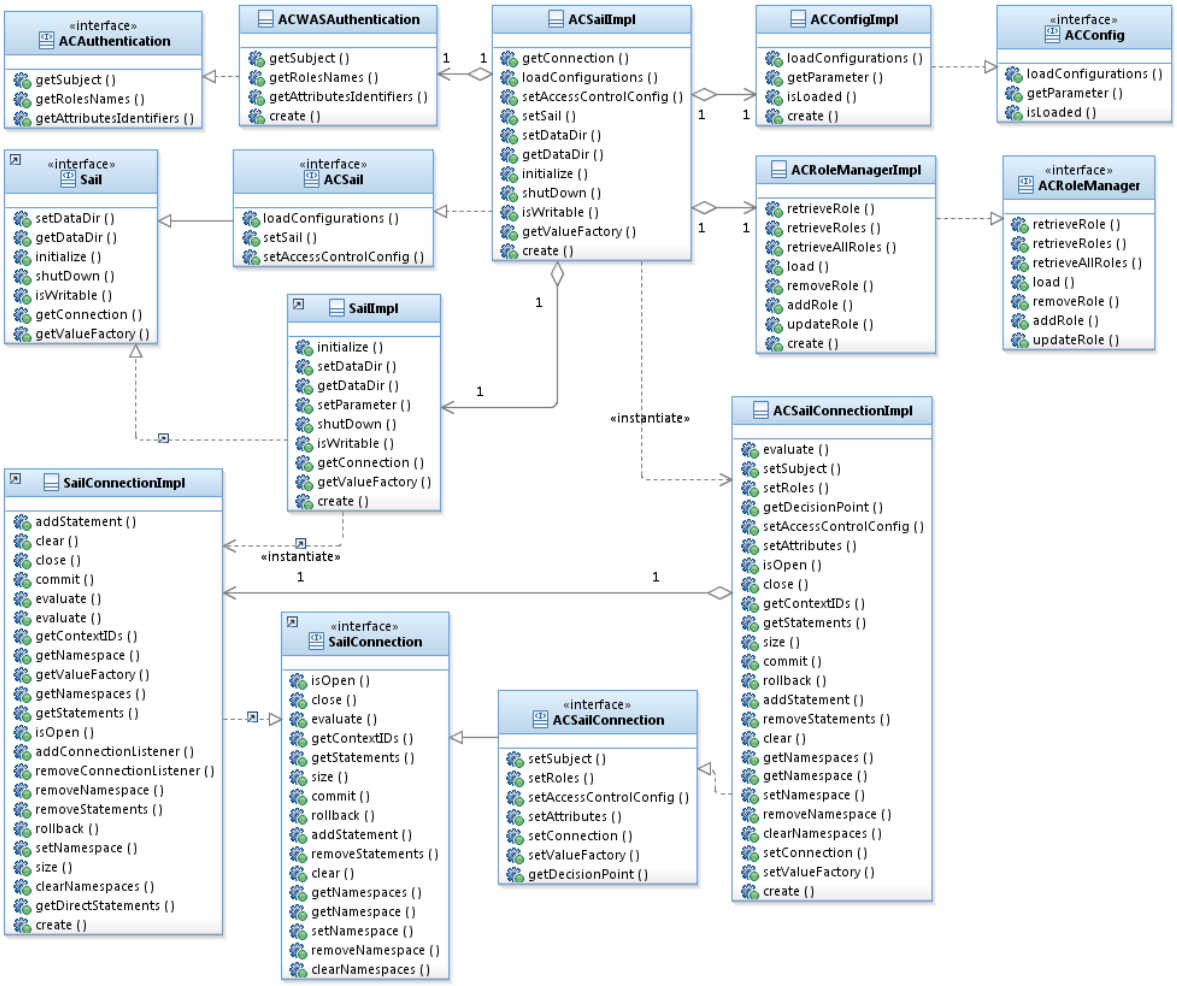


Figura 50 - Diagrama de classes de instanciação de ACSailConnectionImpl

5.1 CRUD

O processo de pesquisa disponibilizado por Sail API (Figura 36) e implementado pelo triple-store SwiftOWLIM funciona como representado no diagrama sequência da Figura 51.

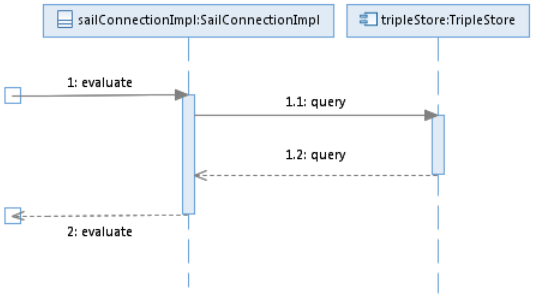


Figura 51 - Pesquisa de SwiftOWLIM

Neste processo SailConnection é implementada por SailConnectionImpl que é responsável por fazer a ligação com o repositório. Contudo esta implementação não permite condicionar o acesso ao repositório e aos factos aí existentes. Para isso é necessário evoluir o processo de

acordo com o processo definido na secção 4.3 (CRUD), como demonstra o próximo diagrama de sequência (Figura 52).

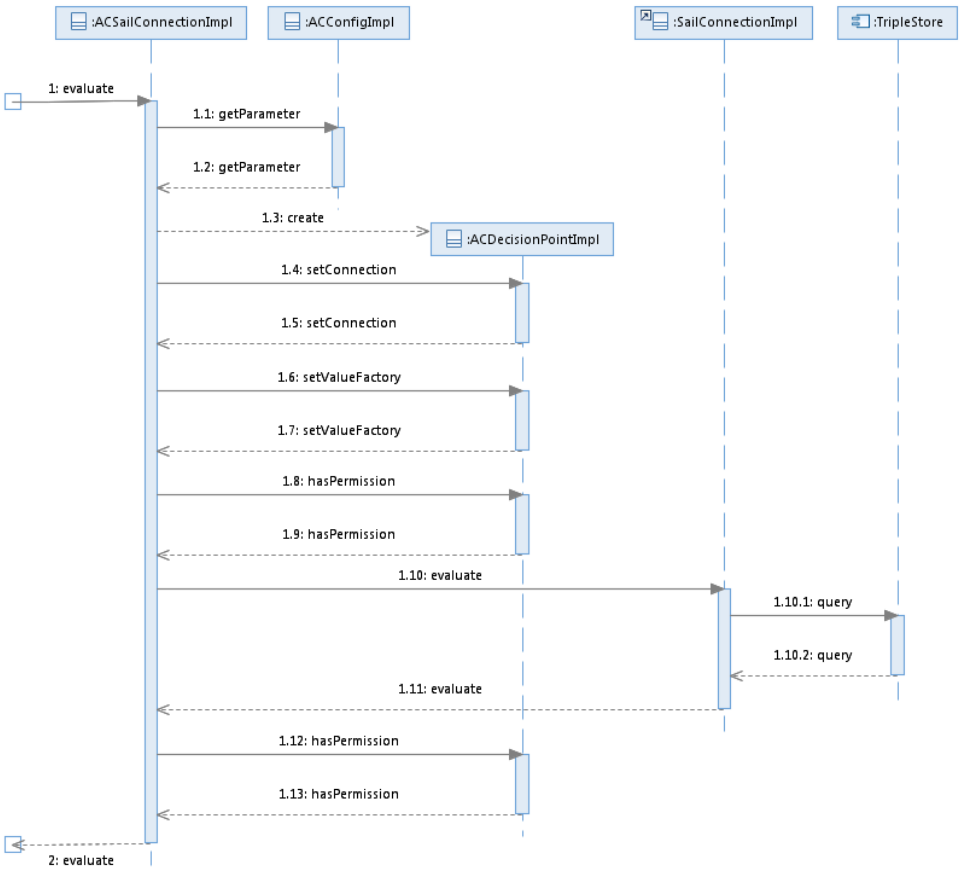


Figura 52 - Pesquisa ACSailConnection através de SwiftOWLIM

De notar que neste processo, em relação ao processo que lhe deu origem, as alterações são unicamente as implementações no lugar das interfaces. Neste processo é de realçar que ACDecisionPointImpl implementa ACDecisionPoint e tem como responsabilidade verificar se o utilizador tem permissões para efectuar a pesquisa. Esta classe implementa o modelo e algoritmo de controlo de acesso descrito na secção 2.1.3.

A Figura 53 representa em notação UML o diagrama de classes aplicadas no diagrama de sequência anterior (Figura 52).

Figura 53 - Diagrama de classes de pesquisa de ACSailConnection através de SwiftOWLIM

5.2 Gestão de papéis

Nesta secção é descrita a implementação duma GUI para a gestão de papéis com um modelo de papéis básico, que explora a interface ACRoleManager introduzida na secção 4.4.

A classe ACRoleManagerImpl é uma implementação da interface ACRoleManager. Esta implementação contém um modelo base de papéis, i.e. os papéis são definidos apenas pelo nome e conjunto de permissões. Este modelo permite qualquer tipo de restrições ou hierarquia entre papéis. Para isso é necessário estender ou criar um novo de modelo de papéis.

Para obter uma implementação de ACRoleManager é necessário carregar o ambiente de execução, tal como demonstra o diagrama de sequência da Figura 54.

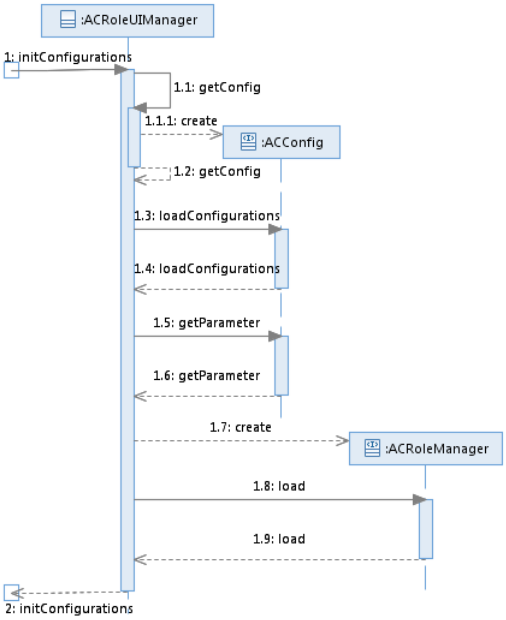


Figura 54 - Carregar ambiente de execução para gerir papéis

A classe ACRoleUIManager pede o carregamento das configurações de execução à implementação da interface ACConfig. A implementação da interface ACConfig é definida no método de instanciação getConfig(). De seguida a classe ACRoleUIManager pede à implementação de ACRoleManager que carregue as suas configurações.

Ao nível da interface com o utilizador foi desenvolvida uma aplicação swing em Java que permite ao utilizador adicionar, consultar, actualizar e remover papéis. Esta GUI está implementada a partir da classe ACRoleUIManager e tem como pressuposto o modelo base de papéis referido anteriormente. Caso seja intenção utilizar outro modelo de papéis deve-se estender a classe mencionada e acrescentar as modificações que forem pertinentes.

A GUI implementada pela classe ACRoleUIManager (Figura 55) delega na interface ACRoleManager a responsabilidade de efectuar as operações de gestão, i.e. ACRoleUIManager quando tem de executar uma operação de criação, leitura, actualização ou eliminação de papéis invoca a implementação da interface ACRoleManager.

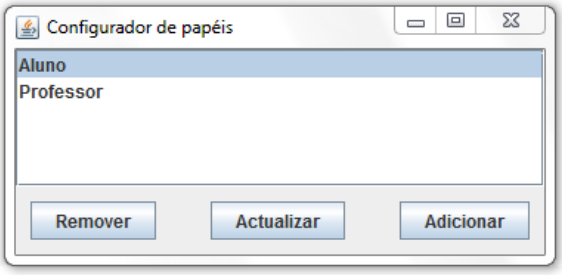


Figura 55 – Screenshot do ecrã inicial de gestão de papéis

Foram criados dois painéis AddRoleDialog (Figura 56) e UpdateRoleDialog (Figura 57) para as operações de criação e actualização de papéis respectivamente.

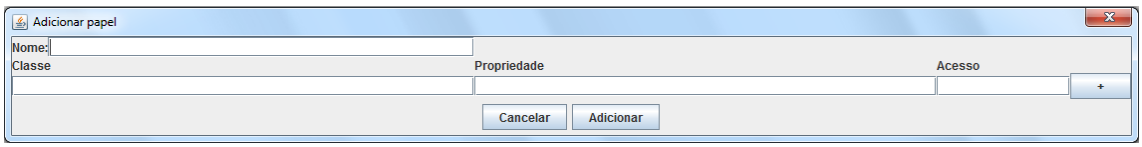


Figura 56 – Screenshot do diálogo de criação de papéis

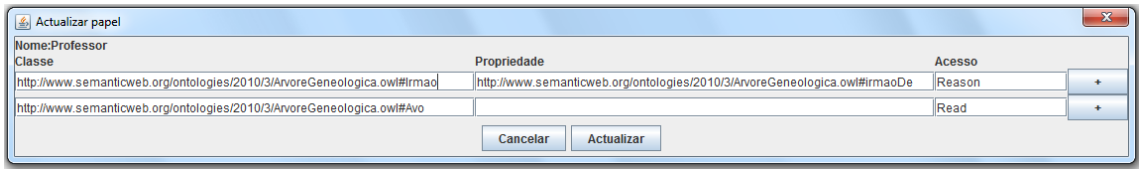
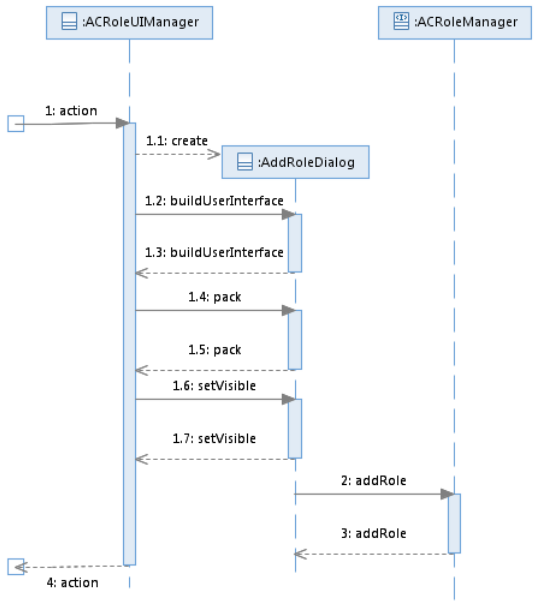


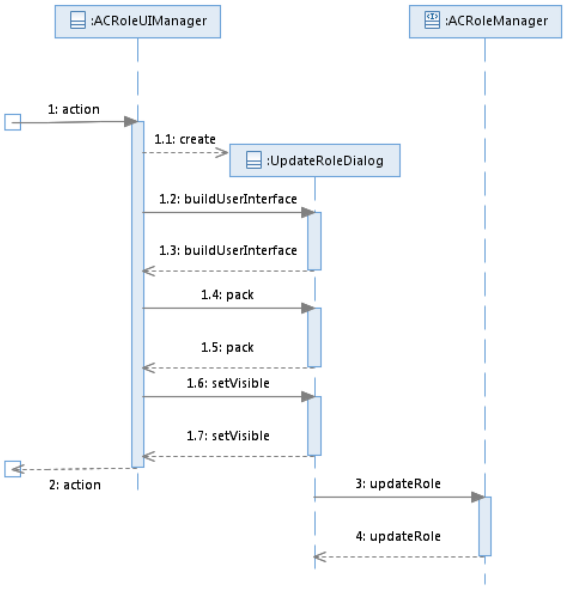
Figura 57 - Screenshot do diálogo de actualização/alteração de papéis

Estes painéis têm por base o modelo de papéis explicado anteriormente. Foram criados de modo a poderem ser estendidos para suportar outros tipos de modelo de papéis, sendo para isso necessário desenvolver classes que estendam das classes AddRoleDialog e UpdateRoleDialog.

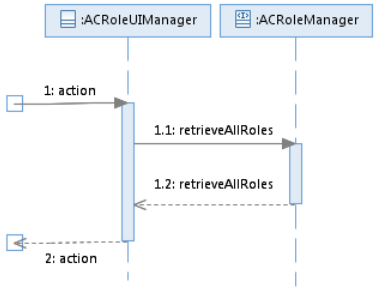
De seguida é preciso informar o sistema que se pretende usar as novas classes, ou seja é necessário estender a classe ACRoleUIManager e nos métodos getAddRoleDialog() e getUpdateRoleDialog() fazer a instanciação das novas classes. A Figura 58 demonstra os diagramas de sequência para as operações de leitura, adição, actualização e eliminação de papéis de ACRoleUIManager.



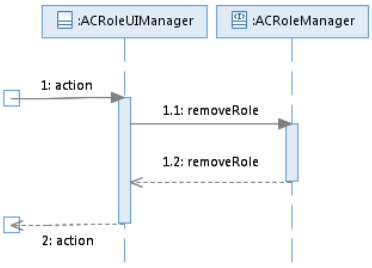
a)



b)



c)



d)

Figura 58 - Adicionar (a) Actualizar (b) Consultar (c) Eliminar (d) papéis

A Figura 59 representa o diagrama UML das classes aplicadas na GUI da aplicação de gestão de papéis.

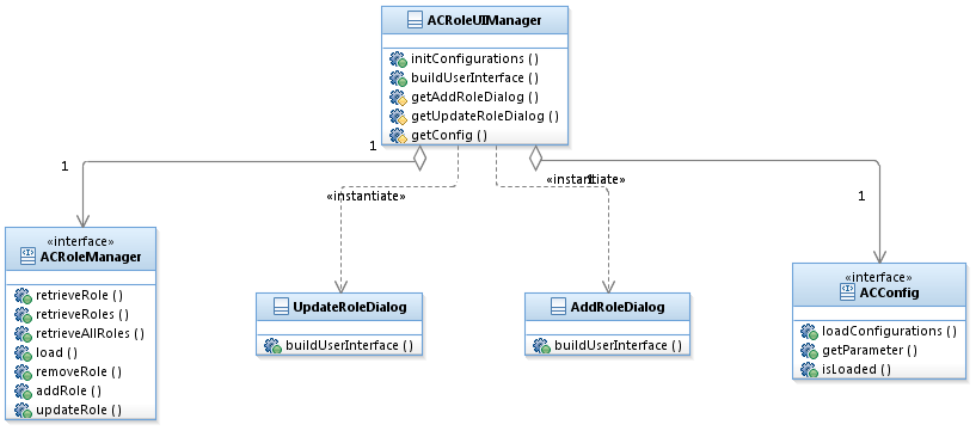


Figura 59 - Diagrama de classes de GUI de gestão de papéis

Como se pode constatar, esta aplicação é meramente demonstrativa da utilização e possibilidades de integração da API proposta no capítulo 4, sendo possível desenvolver novas interfaces com o utilizador.

Capítulo 6

Limitações

O trabalho descrito nos capítulos anteriores tem de ser continuado no que respeita a duas dimensões:

- Testes de integração com outros triple-stores;
- Testes de performances.

6.1 Testes de integração

A framework proposta no Capítulo 4 foi implementada como prova de conceito para o triple-store SwiftOWLIM, cuja descrição foi feita no capítulo Capítulo 5.

Esta implementação foi bem sucedida e o esforço de implementação foi bastante reduzido. A integração da autenticação com WebShere Application Server também foi bem sucedida, e mais uma vez com esforço reduzido.

Contudo, ficam algumas questões por responder:

- Como é que o framework permitiria resolver particularidades de outros triple-stores compatíveis com Sail API;
- Como é que o framework permitiria resolver particularidades de outras aplicações compatíveis com JAAS;
- Qual a opinião que outra equipa de desenvolvimento teria no desenvolvimento e aplicação a outro triple-store e/ou ambiente de autenticação.

As duas primeiras questões dependem evidentemente das particularidades que esses triple-stores e aplicações compatíveis com JAAS tenham. De facto não é suposto que um triple-store compatível com Sail API tenha particularidades que impossibilitem a adopção da framework Access Control, pelo que qualquer esforço no sentido de responder a esta questão enfermam sempre da realização de tal esforço para todos os triple-store e aplicações compatíveis com JAAS.

Já no que respeita à terceira questão, essa requer que a framework seja adoptada e avaliada na prática por outras equipas. Contudo, considerando que foram no seu design adoptados princípios e padrões fundamentais de design Orientado a Objecto (e.g. baixo acoplamento, alta, coesão, padrões GoF, e padrões empresariais), tudo leva a crer que não será muito complicada a sua adopção por outros. Contudo, a prova está por fazer.

6.2 Testes de performance

É pertinente mencionar que os níveis de performance temporal da pesquisa pioram necessariamente com a utilização do modelo de controlo de acesso, mas tal degradação não foi avaliada rigorosamente.

De notar que a degradação de performance do controlo de acesso é fundamentalmente influenciada pelo algoritmo de controlo de acesso aplicado (e que pode variar entre instalações) e neste ponto específico não é possível efectuar nenhuma melhoria a não ser a sua implementação optimizada.

Contudo, e porque é importante minimizar a degradação de performance pela introdução do controlo de acessos, torna-se necessário fazer testes rigorosos que permitam aferir da responsabilidade da nova framework e do algoritmo de controlo de acessos na degradação de performance. Compreender tais responsabilidades permitirá actuar no refinamento da framework, na sua configuração por instalação, e na adopção mais adequada do modelo de controlo de acessos a cada situação.

Capítulo 7

Conclusões e Trabalho Futuro

O trabalho efectuado responde aos objectivos definidos inicialmente:

- permitir controlar acessos a repositórios de triplos descritos por ontologias;
- aplicando diversos e distintos modelos de controlo de acesso;
- integração com ambientes onde a identificação do utilizador já existe, nomeadamente aqueles que aderem à especificação JAAS.

Foi possível criar uma aplicação que ajuda a resolver um dos maiores problemas da semântica Web, i.e. a confidencialidade dos factos. A aplicação não resolve todos os problemas de confidencialidade de factos mas permite resolver, i.e. ao permitir especificar e integrar novos algoritmos de controlo de acesso aos factos faz com que a aplicação consiga ser totalmente abrangente aos algoritmos de controlo de acesso. A resolução do problema definido inicialmente foi efectuada utilizando a Sail API, que é uma API comum a variados triple-stores, e portanto tem desde logo uma significativa abrangência.

Apesar de efectuar controlo de acesso com informações baseadas no sujeito, factos ou no modelo, é possível condicionar o acesso através de informações de outros meios ou repositórios (e.g. informações sobre o clima).

É de realçar que o modelo desenvolvido contém uma integração muito simples com qualquer sistema que contemple autenticação segundo a especificação JAAS, i.e. basta que se desenvolva uma classe que delegue a responsabilidade da autenticação no sistema que respeita a especificação JAAS. Isto é crucial quando se usam servidores aplicacionais como Websphere Application Server, Tomcat, WebLogic, o que potencia a sua aplicação a cenários já existentes e que careciam de solução semelhante à proposta e promovida nesta dissertação.

No que diz respeito ao trabalho futuro, há a salientar que a framework agora proposta e a implementação realizada para SwitOWLIM permitiram criar uma "test-bed" que possibilitará e promoverá a investigação e desenvolvimento neste campo, nomeadamente:

- permitirá avaliar a adequação dos modelos de controlo de acesso a situações particulares, nomeadamente em ambientes abertos e heterogéneos (Web), bem como a ambiente mais controlados (e.g. ambiente empresarial/corporativo);
- desenvolvimento de novos modelos e algoritmos de controlo de acesso que respondam mais convenientemente aos requisitos dos cenários e permitam a sua manutenção de forma mais simples que os actuais.

Referências

Baader, F. et al., 2003. *The Description Logic Handbook: Theory, Implementation and Applications*, Cambridge University Press.

Berners-Lee, T., Hendler, J. & Lassila, O., 2001. The Semantic Web. *Scientific American*, 284(5), p.34—43.

Cirio, L., Cruz, I.F. & Tamassia, R., 2007. A role and attribute based access control system using semantic web technologies. In *Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems - Volume Part II*. OTM'07. Berlin, Heidelberg: Springer-Verlag, pp. 1256–1266. Available at: http://dl.acm.org/citation.cfm?id=1780453.1780518.

Gamma, E. et al., 1994. *Design Patterns: Elements of Reusable Object-Oriented Software* 1st ed., Addison-Wesley Professional.

Grand, M., 2001. *Java Enterprise Design Patterns: Patterns in Java Volume 3*, John Wiley & Sons.

Ionita, C.M. & Osborn, S.L., 2003. Privilege Administration For The Role Graph Model. *IN RESEARCH DIRECTIONS IN DATA AND APPLICATIONS SECURITY, PROC. IFIP WG11.3 WORKING CONFERENCE ON DATABASE SECURITY*, p.15--25.

Ionita, C.M. & Osborn, S.L., 2005. Specifying an Access Control Model for Ontologies for the Semantic Web. In W. Jonker & M. Petković, eds. *Secure Data Management*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 73-85.

Oracle Corporation, 2008. *Developers and Identity Services - Modernizing Access Control with Authorization Service*,

Priebe, T. et al., 2006. Supporting Attribute-based Access Control In Authorization . . . *IN PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON AVAILABILITY, RELIABILITY AND SECURITY (ARES'06), IEEE COMPUTER SOCIETY*, p.465--472.

Sandhu, R. & Sandhu, R., Coyne, E.J., Feinstein, H.L. and Youman, C.E., 1996. Role-Based Access Control Models. *IEEE Computer*, 29(2), pp.38–47.