Controle de Acesso a Dados em Sistemas de Informação através de Mecanismos de Propagação de Identidade e Execução de Regras de Autorização

Felipe Leão, Sergio Puntar, Leonardo Guerreiro Azevedo, Fernanda Baião, Claudia Cappelli

NP^{2}Tec – Núcleo de Pesquisa e Prática em Tecnologia
Departamento de Informática Aplicada
Universidade Federal do Estado do Rio de Janeiro (UNIRIO)

{felipe.leao, sergio.puntar, azevedo, fernanda.baiao, claudia.cappelli}@uniriotec.br

Abstract. *Information security is a critical issue for organizations and comprises several perspectives, including that of information systems and data repositories development. In current distributed information systems architectures, information security involves identity propagation and data access authorization issues. This work describes the design and implementation of a simple, yet not trivial, architecture for effectively assuring information security, based on a series of standard technology.*

Resumo. *Segurança de informação é uma questão crítica em organizações que deve ser tratada sob diferentes perspectivas, incluindo a do desenvolvimento dos sistemas de informação e dos repositórios dos dados acessados por tais sistemas. Nos diversos cenários de arquitetura distribuída dos sistemas de informação atuais, o problema da segurança envolve as questões de propagação de identidade e de regras de autorização de acesso aos dados. Este artigo descreve o projeto e implementação de uma arquitetura eficaz para solução destes problemas que integra, de forma simples e não trivial, uma série de tecnologias disponíveis.*

1. Introdução

Segurança da informação é um dos principais temas discutidos atualmente nas organizações. Dentro dele, aspectos de controle de acesso, controle de interface, auditoria, controle do fluxo de informação, disponibilidade, confidencialidade e avaliação são tratados. Neste universo o aspecto de controle (ou autorização) de acesso, responsável em grande parte pela garantia da integridade [Sandhu *et al.*, 1996; Yang, 2009; Call e Martinenghi, 2008; Murthy e Sedlar, 2007] se apresenta como elemento central. É através de mecanismos que implementam este aspecto, por exemplo, que regras de negócio, declaração que define ou restringe algum aspecto de uma organização [BRG 2000], são garantidas. Regras de negócio têm como objetivo definir a estrutura de um negócio ou controlar ou influenciar o seu comportamento. Em particular, uma categoria de regras de negócio é a assertiva de ação de autorização, ou **regra de autorização**, a qual restringe a **quem** é permitido realizar uma **ação** na organização e sobre quais **informações**.

Em diversos cenários de arquitetura distribuída dos sistemas de informação atuais, onde as camadas de um sistema (aplicação cliente, servidor de aplicação, componente de acesso a dados, servidor de dados) encontram-se implementadas em componentes independentes e alocadas a unidades de processamento distintas, os principais mecanismos para implementação do controle de acesso são a propagação de identidade entre as camadas do sistema de informação e a aplicação de regras de autorização. Estes mecanismos fazem com que a identidade do usuário seja considerada quando do acesso ao dado de modo que sejam manipulados apenas os dados que o usuário tem acesso e também que a aplicação tenha conhecimento do que este determinado usuário pode fazer com a informação acessada.

Este artigo descreve o projeto de uma arquitetura eficaz para implementação de mecanismos de propagação de identidade e execução de regras de autorização, integrando de forma simples, porém não trivial, uma série de tecnologias disponíveis no mercado. O protótipo foi testado quanto a sua eficácia em um ambiente de simulação, tendo obtido resultados satisfatórios.

O trabalho está organizado da seguinte forma. A Seção 2 apresenta 4 cenários de aplicações para acesso a banco de dados, considerados pela solução proposta. A Seção 3 define mecanismos para autorização de acesso em banco de dados. A Seção 4 apresenta a proposta de arquitetura para controle de acesso através de propagação de identidade. A Seção 5 apresenta o projeto e a implementação da proposta, enquanto que a Seção 6 apresenta os testes experimentais realizados. Finalmente, a Seção 7 conclui o trabalho.

2. Cenários de arquitetura distribuída de sistemas de informação

Esta Seção descreve quatro cenários genéricos de arquiteturas de sistemas de informação em ambientes distribuídos, freqüentemente encontrados nas organizações.

Cliente acessa banco diretamente executando um processo identificável. Este cenário contempla o conjunto de aplicações que, através de processos executados no computador do cliente, fazem acesso direto a base de dados para o consumo de informações e processamento local das mesmas. Estes processos são identificáveis porque são executados a partir do usuário real da aplicação, que é o usuário do sistema operacional. O banco é acessado através de uma conexão direta ao banco de dados, por exemplo, conexão JDBC,[1] que utiliza um usuário de banco de dados único para cada aplicação. Entretanto, é possível capturar parâmetros de conexão como o usuário do sistema operacional e, portanto, o usuário real diretamente a partir de variáveis do SGBD (Sistema Gerenciador de Banco de Dados) que controla o acesso ao banco de dados. Um exemplo é o atributo OS_USER existente no *namespace userenv*[2] no SGBD Oracle. Este cenário contempla aplicações cliente-servidor, mas também aplicações clientes do banco de dados como o SQL Plus[3], por exemplo.

Cliente acessa o banco diretamente executando um processo não identificável. A diferença do segundo cenário tecnológico em relação ao primeiro está em como o processo[4] que consome e processa informações da base de dados é executado. Este

[1] http://java.sun.com/products/jdbc/overview.html
[2] Userenv é um namespace do SGBD Oracle para acesso a informações da sessão de um cliente.
[3] http://www.oracle.com/technology/docs/tech/sql_plus/index.html
[4] Entende-se por processo um executável ou serviço rodando no sistema operacional que acessa o banco de dados.

processo executado no cliente é iniciado com um usuário do processo distinto do usuário do sistema operacional. Por exemplo, um processo *P* executando no sistema operacional que acessa o banco de dados é executado por um usuário privilegiado do sistema operacional chamado *UsrP* ao invés do usuário logado *UsrL*. Portanto, não é trivial a identificação do usuário da aplicação a partir da conexão com o banco advinda destes processos: na abordagem anterior, o parâmetro OS_USER traria o usuário genérico da aplicação para acesso ao banco. Isso impossibilitaria, o processo de autorização dos dados por usuário por desconhecer quem é o usuário real da aplicação.

Cliente acessa servidor de aplicação que acessa o banco. No terceiro cenário, está o conjunto de aplicações que possuem um cliente que deve se conectar a um servidor de aplicação que, por sua vez, acessa a base de dados. Neste cenário, o servidor acessa a base de dados através de um usuário genérico no banco de dados, não sendo uma tarefa simples identificar o usuário que executou a aplicação cliente.

Cliente acessa servidor de aplicação via serviços web. No quarto cenário a arquitetura da aplicação é composta por um servidor de aplicação que acessa a base de dados através de um ou mais usuários criados por instância da base de dados. Em posse desses dados, esse servidor os expõe através de serviços web. Neste cenário, os clientes que acessam este servidor de aplicação podem ser outras aplicações, o que deixaria o usuário real em uma camada ainda mais distante.

3. Mecanismos para controle de autorização de acesso a dados

Existem diferentes mecanismos de controle de autorização de acesso a dados, os quais podem ser divididos em DAC (*Discretionary Access Control*), MAC (*Mandatory Access Control*), ambos propostos por [DoD 1983], e mais recentemente RBAC (*Role-Based Access Control*) [Ferraiolo e Khun, 1992].

O mecanismo DAC restringe acesso a objetos baseado na identidade de usuários e/ou grupos aos quais eles pertencem. [Yang 2009] aponta que políticas DAC não garantem controle sobre o fluxo de informações, pois torna possível que informações cheguem a usuários que não têm permissão de lê-las. Políticas MAC, também conhecidas como *label security*, são baseadas em regulamentações mandatórias determinadas por uma autoridade central [Yang 2009]. A forma mais comum de MAC é a política de segurança de múltiplos níveis usando classificação de sujeitos (usuários ou grupos) e objetos (dados) dos sistemas. MAC controla o fluxo de informações, embora não aborde as ações que podem ser executadas pelos sujeitos sobre os dados [Ferraiolo e Khun, 1992]. Além disso, como um rótulo é atribuído a cada instância de dados, o custo de gestão e de manutenção dos rótulos é alto, e pouco flexível, se existirem muitas políticas definidas. No caso de RBAC, o controle de acesso considera funções e informações, além das informações. Neste caso, o interesse principal é proteger a integridade da informação: quem pode realizar qual ação sobre qual informação [Ferraiolo e Khun, 1992]. Em [Ferraiolo *et al*. 2001] é proposto um padrão para RBAC a fim de unificar idéias existentes em diferentes modelos de referência de RBAC, produtos comerciais e protótipos de pesquisa.

Os mecanismos para controle de acesso encontram-se implementados em diferentes Sistemas Gerenciadores de Banco de Dados (SGBD), como Oracle, Sybase, Microsoft SQL-Server [Yang 2009]. Portanto, o acesso dos usuários utilizando as

aplicações disponibilizadas pelas organizações poderia ser controlado diretamente pelos mecanismos de controle de acesso existentes nos SGBDs. No entanto, existem diferentes cenários de conexão das aplicações com os SGBDs, como detalhado na Seção 2. Para alguns cenários o uso das funcionalidades dos SGBDs é simples, sendo fácil identificar o usuário que está logado na aplicação e, a partir daí, aplicar as regras que se referem ao mesmo. Por outro lado, em outros cenários não é simples identificar o usuário diretamente, sendo necessário utilizar outros mecanismos para que o SGBD consiga aplicar as regras de autorização sobre o usuário utilizando a aplicação.

4. Uma Arquitetura Prática e Eficaz para o Aspecto de Controle de Acesso

Nos cenários de arquitetura de Sistemas de Informação descritos na Seção 2, a solução para o aspecto de controle de acesso passa por implementar mecanismos relacionados à definição dos perfis e das regras de autorização; à autenticação do usuário; à propagação de identidade do usuário autenticado entre as camadas do sistema até o servidor de dados e, finalmente, uma vez que o usuário atual do sistema de informação tenha sido reconhecido pelo SGBD, à execução das regras de autorização em tempo de execução. Esta Seção apresenta a arquitetura para controle de acesso proposta, descrevendo em detalhes os aspectos de execução das regras de autorização (Seção 4.1) e de propagação de identidade (Seção 4.2). A implementação de mecanismos relacionados à definição de perfis e regras de autorização foi tratada em Azevedo *et al*. [2010]. A implementação de mecanismos de autenticação do usuário está fora do escopo deste trabalho.

4.1 Mecanismos para execução de regras de autorização de acesso a dados

A solução da arquitetura proposta neste trabalho para execução de regras de autorização de acesso a dados considera o uso do mecanismo de controle de acesso RBAC implementado no SGBD Oracle empregando a proposta de modelo flexível para controle de acesso proposta por Azevedo *et al.* [2010], denominada FARBAC (*Flexible Framework for Role-Based Access Control*). Este modelo foi implementado utilizando o mecanismo VPD – *Virtual Private Database* [Jeloka *et al.*, 2008] presente no SGBD (Sistema Gerenciador de Banco de Dados) Oracle.

No FARBAC as regras de autorização são definidas como cláusulas WHERE, que são dinamicamente adicionadas aos comandos enviados pelas aplicações ao SGBD. Em seleções, por exemplo, as regras funcionam como um filtro, removendo da consulta os registros que o usuário não possui acesso. As regras são associadas a perfis (ou papéis) que são concedidos a cada usuário. O FARBAC recupera em tempo de execução o usuário executando o comando e aplica as regras referentes aos perfis desse usuário.

Uma premissa para o funcionamento do FARBAC é que a identidade do usuário seja propagada corretamente nas diversas camadas do sistema. Na atual implementação do modelo no Oracle, a identidade do usuário chega ao SGBD através da chamada ao procedimento *dbms_application_info.set_client_info*, o qual define um valor para o atributo *client_info* do *namespace userenv*, posteriormente recuperado pelo FARBAC.

4.1.1 Modelo conceitual das regras de autorização

As regras de autorização do FARBAC são armazenadas no banco de dados de regras de negócio de acordo com o modelo conceitual de entidades e relacionamentos apresentado na Figura 1 e descrito a seguir.

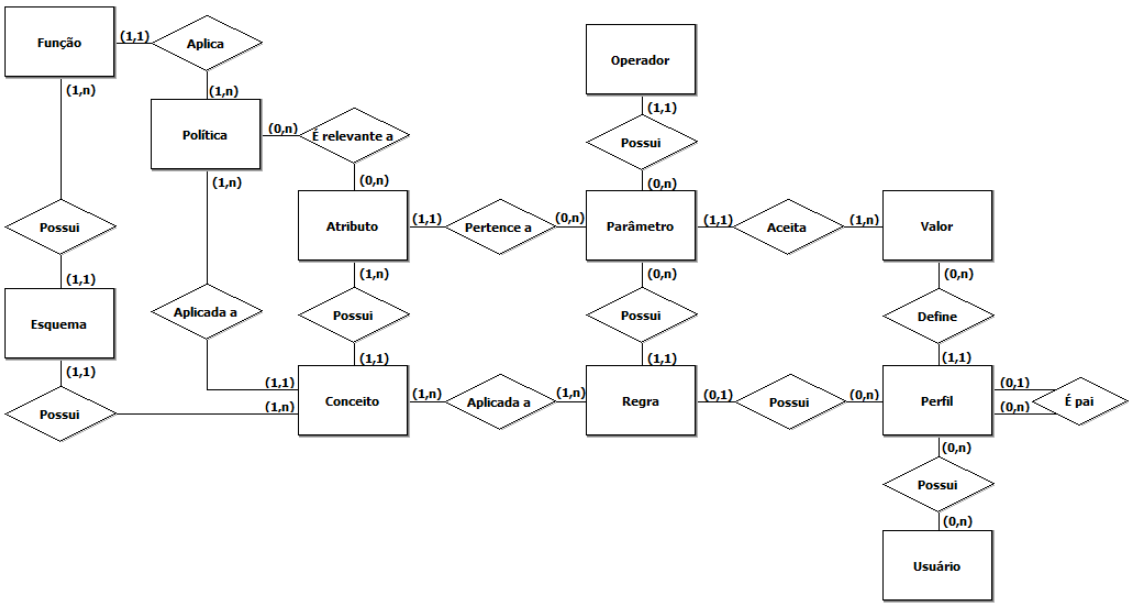

Figura 1 - Modelo de entidade e relacionamento do FARBAC.

Os **Usuários** são agrupados em **Perfis**, onde cada usuário pode ser associado a diversos perfis, e cada perfil pode agrupar diversos usuários. O auto-relacionamento da entidade **Perfil** permite a representação de uma hierarquia de perfis. Por exemplo, os perfis de Gerente de Vendas da América e Ásia e de Gerente de Vendas da Europa podem ser especializações do perfil Gerente de Vendas. Isso significa que regras de autorização atribuídas a um perfil mais geral (Gerente de Vendas) devem também ser aplicadas a todos os seus descendentes. Além disso, perfis descendentes podem determinar valores específicos para os parâmetros do predicado. Por exemplo, o perfil Gerente de Vendas pode restringir o acesso aos pedidos baseado no atributo continenteDeOrigem; o perfil Gerente de Vendas da América e Ásia pode então especificar o(s) valor(es) para o atributo controlado (por exemplo, "America, Asia"). Dessa forma é possível que a política RBAC implemente a regra, restringindo o acesso do perfil Gerente de Vendas América e Ásia somente para os pedidos onde "*continenteDeOrigem in ('America', 'Asia')*".

A entidade **Regra** representa as definições de regras que são armazenadas como cláusulas SQL, que serão usadas para transformar os comandos SQL provenientes da aplicação cliente. Uma cláusula restringe o acesso do **Usuário** a um subconjunto dos dados de um **Conceito**, que pode ser uma tabela, uma visão ou um sinônimo. Portanto, existe um relacionamento **Regra** × **Conceito**. A mesma cláusula SQL pode ser usada para definir regras de autorização em diferentes conceitos, e cada **Conceito** pode possuir mais de uma **Regra**.

Um **Parâmetro** denota uma expressão no formato "Atributo Operador Valor" relacionando as entidades correspondentes (**Atributo** × **Operador** × **Valor**). A entidade **Atributo** representa um atributo de um conceito do banco de dados utilizado para restrição de acesso. A entidade **Operador** representa um operador binário (incluindo =, <>, >, >=, <, <=, IN e NOT IN). A entidade **Valor** representa um valor (ou lista de valores) dentro do domínio do atributo, que delimitam um subconjunto de dados. O relacionamento **Conceito** × **Atributo** indica de que conceito um atributo pertence da

mesma forma que o relacionamento **Esquema** × **Conceito** indica de que esquema o conceito pertence. Esse relacionamento denota uma representação em alto nível do esquema do banco de dados para o qual as regras de autorização são aplicadas.

A **Política** dita o controle de acesso que deve ser aplicado a um conceito. Uma política de controle de acesso é aplicada por uma **Função** (que também pertence a um **Esquema**). Durante sua execução, esta função identifica o *Conceito* sendo acessado e o *Usuário* executando o comando. Ela captura os dados das outras tabelas do modelo para construir o predicado da regra de autorização (ou predicado de autorização) daquele conceito para aquele usuário. Além disso, o relacionamento **Política** × **Atributo** determina quais atributos do conceito possuem dados sensíveis.

O mesmo usuário pode possuir mais de um perfil relacionado a um mesmo conceito. Dessa forma, em tempo de execução, mais de uma regra (predicado de autorização) será retornada e, portanto, devem ser compostas de alguma forma. Essa composição pode ser feita através do uso dos operadores OR ou AND. A composição por OR, que chamamos de composição permissiva, permite que o usuário tenha acesso à união dos subconjuntos de dados permitidos por cada regra. A composição por AND, que chamamos de composição restritiva, limita o acesso do usuário à interseção dos subconjuntos de dados permitidos por cada regra. Por exemplo, um usuário que possua os perfis Gerente de Vendas da Europa e Gerente de Vendas da França, com a composição permissiva teria acesso aos dados de toda a Europa, já com a composição restritiva o seu acesso seria apenas aos dados da França. Tanto a composição permissiva quanto a composição restritiva são suportadas pelo FARBAC, e é dever da organização decidir que tipo usar.

4.1.2 Algoritmo para construção do predicado de autorização

O FARBAC possui uma função genérica responsável por construir, em tempo de execução, o predicado de autorização para cada conceito. O predicado de autorização é a composição (permissiva ou restritiva) de todas as regras de um usuário para um conceito. A função é aplicada a todos os conceitos com dados sensíveis e, de acordo com o usuário realizando o acesso e com os dados cadastrados no modelo de armazenamento de regras de autorização (Figura 2), retorna o predicado de autorização.

```
Recupera o usuário que está executando a consulta
A partir do relacionamento Usuário × Perfil, recupera os perfis do usuário
Para cada perfil Faça
    A partir do relacionamento Perfil × Regra × Conceito, identifica as regras
    que devem ser aplicadas de acordo com o perfil pai da hierarquia
    Para cada regra Faça
        A partir do relacionamento Regra × Parâmetro, identifica os parâmetros
        específicos da regra
        Para cada parâmetro Faça
            A partir do relacionamento Perfil × Valor, recuperar o(s) valor(s)
            aceito(s) pelo parâmetro para filtragem dos dados
Finalmente, o predicado de autorização é montado e retornado
```

Figura 2 - Algoritmo de construção do predicado de autorização.

4.2 Mecanismos para propagação de identidade

A propagação de identidade procura permitir que a identidade do usuário seja preservada, independente de onde a informação da identidade foi criada, para utilização durante autorização e para fins de auditoria [IBM, 2010]. A proposta para propagação de

identidade é utilizar o padrão de injeção de dependência (do inglês *Dependency Injection*) o qual é um padrão de projeto derivado do padrão Inversão de Controle, e assim como seu progenitor visa fornecer a uma classe informações necessárias para o seu funcionamento sem que a classe tenha que se preocupar com a geração destes dados. Neste caso, propomos o uso de injeção via *setter* (*Setter Injection*), a qual consiste em passar as dependências através de argumentos por um método *setter* [Prasanna, 2009].

Outros mecanismos para propagação de identidade foram analisados como, por exemplo, o uso de arquivos de configuração para definir injeção de dependência em tempo de execução. As informações do usuário são injetadas em objetos que executam a lógica do negócio. Foram analisadas as propostas de implementação utilizando EJB 3.0 e Spring. Como resultado, observamos que o uso destas propostas não permite injetar um objeto considerando exatamente o cliente que está invocando a operação do objeto que implementa a lógica do negócio. Ou seja, no objeto remoto não se consegue identificar o cliente para a injeção de dependência, pois a configuração é estática. É permitido apenas injetar um objeto genérico para todas as chamadas do objeto remoto.

4.3 Arquitetura para propagação de identidade e autorização de acesso

A proposta de arquitetura para propagação de identidade e execução de regras de autorização é ilustrada na Figura 3. É importante enfatizar que a proposta considera que o usuário do sistema de informação está devidamente autenticado para a propagação de sua identidade. Segundo Needham e Schroeder [1978], autenticação corresponde à verificação mútua da identidade das partes que estão se comunicando.

Tendo em mãos as informações do usuário autenticado, a aplicação cliente repassa estas informações para as camadas seguintes até chegar ao SGBD. O SGBD, por sua vez, implementa a arquitetura do FARBAC (descrita na Seção 4.1). Na Figura 3, é feita a relação entre os cenários apresentados na Seção 2 e a proposta: (i) cenários 1 e 2 - aplicação acessando o banco de dados diretamente: as informações do usuário são passadas diretamente da aplicação para o SGBD para realizar acesso controlado ao banco de dados; (ii) cenário 3: aplicação cliente passa as informações do usuário para o servidor de aplicação que passa para a mesma para o SGBD; (iii) cenário 4 – múltiplas camadas: aplicação cliente passa as informações do usuário para um servidor de aplicação, no qual existem serviços web executando, e este repassa ao servidor de aplicação ou, caso o serviço web acesse o banco de dados, as informações são passadas diretamente para o SGBD.

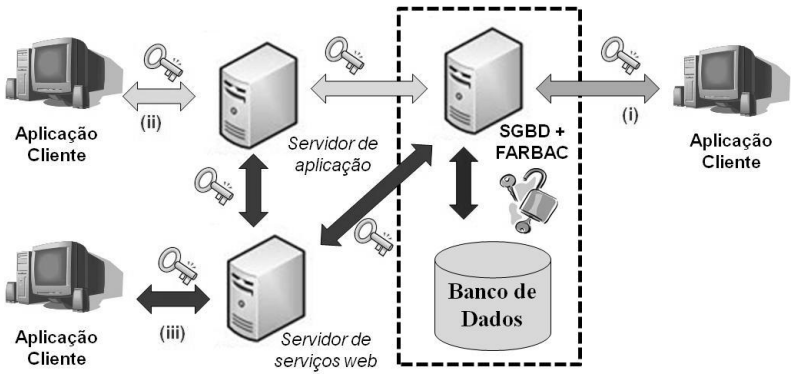

Figura 3 – Propagação de identidade nos cenários da arquitetura proposta.

5. Projeto e Implementação da Arquitetura Proposta

A implementação de mecanismos para propagação de identidade do usuário segundo a arquitetura proposta contempla 3 dos 4 cenários descritos na Seção 2 e emprega tecnologias de amplo uso no mercado e padrões de programação.

Um protótipo foi implementado utilizando as tecnologias Java Enterprise Edition e Standart Edition, Enterprise Java Beans 3.0 (EJB3), servidor de aplicação JBoss (versão 4.2), *framework* de mapeamento objeto-relacional Hibernate (versão 3), SGBD Oracle (versão 10g) e os padrões de projeto *Service Locator* e *Dependency Injection*. O banco de dados utilizados para os testes seguiu o esquema do *benchmark* TPC-H [TPC Council, 2008], o qual consiste em uma especificação de grande relevância na indústria que simula um cenário de uma aplicação de apoio à decisão. O protótipo implementado trata os primeiros três cenários apresentados na Seção 2. Para o quarto cenário, estão sendo avaliados o uso de padrões de web services, tais como, SOAP[5], WS-Security[6] e WS-Policy[7] para transportar de forma segura as informações do usuário no cabeçalho da mensagem SOAP. No entanto, vale ressaltar que os aspectos de propagação de identidade apresentados neste trabalho são os mesmos que estão sendo avaliados para o uso destas tecnologias.

O protótipo (Figura 4) consiste em uma aplicação cliente (Java SE), que acessa um EJB *Stateful* responsável por realizar operações de consulta ao banco de dados Oracle e devolver o resultado da pesquisa à aplicação cliente. O acesso ao EJB é realizado através do padrão de projeto *Service Locator*. O EJB acessa o SGBD através do framework de mapeamento objeto-relacional Hibernate.

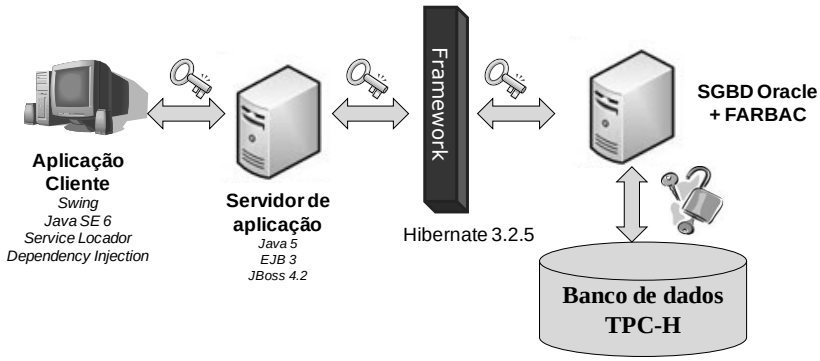

Figura 4 - Arquitetura do protótipo.

5.1. Aplicação Cliente

Na arquitetura do protótipo implementado, a aplicação cliente utiliza a tecnologia Java *Swing* para solicitar a execução de uma consulta no banco de dados. O passo a passo de execução de uma requisição na aplicação Cliente é apresentado de forma geral a seguir. Além disso, o diagrama de sequência com maiores detalhes deste passo a passo é apresentado na página do projeto[8].

[5] http://www.w3.org/TR/soap/
[6] http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf
[7] http://www.w3.org/TR/ws-policy/
[8] http://pdac.googlecode.com/

1. No módulo cliente, o usuário informa a chave para acessar o sistema[9].
2. O módulo cliente armazena a chave do usuário na classe *ServiceLocator*.
3. O usuário acessa a funcionalidade implementada.
4. A aplicação cliente acessa o servidor através do *ServiceLocator* para recuperar um componente EJB passando nome JNDI[10] do EJB.
5. A classe *ServiceLocator* verifica se o EJB recuperado implementa a interface *ControleAcesso* (explicada na Seção 5.2); em caso positivo, invoca o método "*setUsuario*" passando a chave armazenada e devolve o EJB à aplicação.
6. A aplicação invoca o método de consulta do componente EJB.
7. A aplicação recebe o resultado da consulta.
8. O módulo cliente apresenta as informações ao usuário.

5.2. Servidor de aplicação

No servidor de aplicação, as regras de negócio estão implementadas em projetos Java utilizando a tecnologia EJB (*Stateful*). Neste caso, temos uma classe Java EJB para cada projeto EJB. O acesso e persistência de dados foi implementado utilizando o framework de mapeamento objeto relacional Hibernate[11], acessando SGBD Oracle. A responsabilidade de gerenciar o ciclo de vida das conexões com o banco de dados (*pool* de conexão) e o contexto transacional é do servidor de aplicação JBoss[12].

Uma interface, chamada *ControleAcesso*, foi implementada para especificar os métodos que os EJB devem implementar para armazenamento das informações necessárias para controle de acesso aos dados. Esta interface é apresentada na Figura 5.

```
1  public interface ControleAcesso {
2      void setUsuario(String chaveUsuario);
3      String getUsuario();
4  }
```

Figura 5 - Implementação da Interface *ControleAcesso.*

A interface *ControleAcesso* é parte de uma biblioteca independente, devendo ser importada para o *classpath* de cada projeto EJB antes de ser utilizada. Esta separação permite que posteriormente outros projetos possam utilizar a mesma interface ao implementar controle de acesso aos dados.

Em cada projeto EJB, é criada uma interface com o objetivo de expor os métodos que poderão ser acessados pelo cliente. Esta interface deve herdar a interface *ControleAcesso*, obrigando o EJB a implementar também os métodos *setUsuario* e *getUsuario* de *ControleAcesso*. O método *setUsuario* é utilizado pela aplicação cliente para ajustar no EJB as informações do usuário, de acordo com o padrão de projeto Injeção de Dependência, cujo uso é proposto na arquitetura.

O acesso ao banco de dados foi implementado utilizando o padrão *HibernateUtil*[13] sugerido pelos desenvolvedores do framework Hibernate, centralizando

[9] A passagem da chave foi suficiente para os testes realizados. Não foi feita validação de senha, pois autenticação ficou fora do escopo do trabalho

[10] JNDI (*Java Naming and Directory Interface*) é uma API para descobrir e acessar recursos externos a aplicações Java. Maiores detalhes www.oracle.com/technetwork/java/jndi/index.html.

[11] http://www.hibernate.org/

[12] O servidor de aplicação utilizado é o JBoss 4.2

[13] A implementação base para a classe HibernateUtil é demonstrada na documentação do Hibernate e pode ser vista em http://community.jboss.org/wiki/sessionsandtransactions.

na aplicação a inicialização do Hibernate e o *lookup* à fábrica de sessões com o banco de dados. Foi considerada uma modificação nesta classe, onde, ao invés de a aplicação requisitar ao *HibernateUtil* uma fábrica de sessões, ela deverá requisitar diretamente uma sessão. O objetivo é informar ao SGBD o usuário que irá realizar as operações no banco de dados. A Figura 6 apresenta a implementação do método *getSession*, da classe *HibernateUtil*, onde é criada uma sessão com o banco de dados e armazenam-se as informações do usuário neste banco antes de retornar o objeto de sessão para quem invocou o método. Neste caso, como o SGBD utilizado é o Oracle, o armazenamento das informações do usuário é feito através da chamada ao procedimento *dbms_application_info.set_client_info*. Esta chamada armazena a chave do usuário no atributo *client_info* do *namespace userenv* do contexto do banco de dados Oracle.

```
33      public Session getSession(String chaveUsuario) {
34          if (session.get() == null) {
35              session.set(sessionFactory.openSession());
36          }
37
38          //preparar o contexto da sessao no banco com a chave do usuario
39          Connection con = session.get().connection();
40          try{
41              CallableStatement st = con.prepareCall(
42                      "{call dbms_application_info.set_client_info(?)}");
43              st.setString(1, chaveUsuario);
44              st.executeUpdate();
45          }catch(SQLException e){
46              return null;
47          }
48
49          return session.get();
50      }
```

Figura 6 - Método getSession da classe HibernateUtil.

O passo a passo de atendimento a uma requisição pelo EJB também se encontra disponível, através de diagrama de sequência, na página web do projeto, sendo descrito em linhas gerais a seguir:

1. O EJB instancia um objeto DAO e armazena nele a chave do usuário recebida.
2. O EJB invoca o método do objeto DAO responsável por executar a operação no BD.
3. O objeto DAO recupera a instância única da classe HibernateUtil e invoca o método *getSession* informando a chave do usuário.
4. A classe HibernateUtil cria uma sessão com o banco de dados e ajusta o usuário que está fazendo uso da sessão através da chamada ao procedimento *dbms_application_info.set_client_info*, então retorna o objeto de sessão.
5. A classe DAO executa a consulta ao banco de dados utilizando sessão retornada. Nesse momento, o FARBAC entra em ação para aplicar a regra de autorização e retornar apenas as informações que o usuário tem acesso.
6. O resultado da consulta é repassado para o EJB que repassa para a aplicação cliente.

6. Testes experimentais

Para avaliar a arquitetura proposta, foi definida uma regra de autorização sobre o esquema do *benchmark* TPC-H, O TPC-H é uma especificação de *benchmark* com uma ampla relevância na indústria, que simula o cenário de uma aplicação de apoio à decisão [TPC *Council*, 2008]. Esse cenário possui um contexto realista que representa as atividades de uma indústria de atacado que precisa gerenciar as suas vendas ou distribuir um produto em todo o mundo (por exemplo, aluguel de veículos, distribuição de

alimentos, etc.). O *benchmark* consiste da descrição de um esquema de banco de dados e de um conjunto de consultas *ad-hoc* OLAP (*Online Analytical Processing*) orientadas ao negócio. A consulta utilizada no teste foi a de previsão de mudança na receita, a qual quantifica o aumento da receita resultante da eliminação de algum desconto percentual em determinado ano (Figura).

```
select sum(l_extendedprice * l_discount) as revenue
from lineitem
where l_shipdate >= date '1994-01-01'
  and l_shipdate < date '1994-01-01' + interval '1' year
  and l_discount between 0.06 - 0.01 and 0.06 + 0.01
  and l_quantity < 24;
```

Figura 7 – Previsão de mudança na receita

A regra de autorização implementada no FARBAC foi: "O gerente de vendas do norte da América e da Ásia deve ter acesso somente aos pedidos provenientes de fornecedores das nações do hemisfério Norte das regiões Ásia e América". A implementação desta regra necessitou de algumas adaptações no modelo do TPC-H e no FARBAC, tais como: a chave do usuário na tabela *CUSTORMER* do TPCH referenciado a chave do usuário do FARBAC; inclusão de campo para nação do usuário no FARBAC; e, inclusão de campo para indicar o hemisfério do usuário.

Para implementar essa regra de autorização foi definida uma hierarquia de perfis, com um perfil pai chamado *Gerente de Vendas* e um perfil filho chamado *Gerente de Vendas Norte América e Ásia*. As informações que devem ser protegidas por essa política são armazenadas nas tabelas *ORDERS* e *LINEITEM*, portanto, foi criado um predicado para cada uma dessas tabelas.

Foram criados os usuários: *Bob* como gerente de vendas norte da América e da Ásia; e *Alice*, que é a presidente da empresa e recebeu o privilégio *EXEMPT ACCESS POLICY*, que faz com que todas as políticas sejam ignoradas.

Em seguida, a Aplicação Cliente foi executada para invocar a consulta apresentada na Figura 7, com os usuário *Bob* e *Alice*. Como *Bob* é o gerente de vendas norte da América e da Ásia, ele tem acesso apenas aos itens de pedido do norte dessas regiões, enquanto *Alice*, que é a presidente, tem acesso a todos os itens de pedido. Para *Alice*, a mudança na receita foi de R$ *88.894.386,60*. Já para *Bob* esta é de apenas R$ *21.445.915,50*, afinal a sua visão é somente dos itens de pedido da América e da Ásia

7. Conclusões

Segurança da informação é uma questão importante para as organizações. Em geral questões desta área são resolvidas através da implementação de mecanismos de controle de acesso e da implementação de regras de autorização em Sistemas de Informação. Contudo, quando uma regra muda, todos estes Sistemas que implementam estas regras e controles devem ser atualizados, fazendo com que isso se torne um problema bastante complexo principalmente em cenários onde existem sistemas legados e um número muito grande de regras de autorização.

Este trabalho apresentou uma proposta e implementação de uma arquitetura de controle de acesso através do uso de mecanismos de propagação de identidade e de execução de regras de autorização de informação, considerando um ambiente que já possua soluções para autenticação dos usuários. A implementação empregou uma série

de tecnologias disponíveis e de amplo uso no mercado, fazendo com que possa ser reutilizada em outros cenários. Testes experimentais foram realizados demonstrando a viabilidade e eficácia da proposta em um cenário que simula uma aplicação real de apoio à decisão.

Como trabalhos futuros, propomos a realização de testes de desempenho e avaliação em um cenário real de uma empresa ou órgão governamental, bem como evolução da solução para contemplar o quarto cenário apresentado na Seção 2.

Referências Bibliográficas

Azevedo, L. G., Puntar, S., Thiago, R., Baião, F., Cappelli, C (2010) "A Flexible Framework for Applying Data Access Authorization Business Rules"**.** In: 12th International Conference on Enterprise Information Systems, pp. 275-280.

BRG, 2000. *Defining Business Rules ~ What Are They Really?*, Rev. 1.3, 2000, http://www.businessrulesgroup.org/first_paper/BRG-whatisBR_3ed.pdf.

Call, A., Martinenghi, D. (2008). "Querying data under access limitations". In: IEEE 24th International Conference on Data Engineering, pp. 50 – 59.

DoD (1983) "Trusted Computer Security Evaluation Criteria". Department of Defense, DoD 5200.28-STD.

Ferraiolo, D.F. e Khun, D. R. (1992) "Role-Based Access Control". In: 15th National Computer Security Conference, pp. 554—563, Baltimore, MD, 1992.

Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R. (2001) "Proposed NIST standard for role-based access control". ACM Transactions on Information and System Security (TISSEC) 4 (3), pp. 224—274,.

IBM (2010) RACF Security Guide. IBM, SC34-7003-01.

Jeloka, S., Mulagund, G., Lewis N. *et al*. (2008). "Oracle Database Security Guide, Oracle RDBMS 10gR2". Oracle Corporation. http://download.oracle.com/docs/cd/B19306_01/network.102/b14266.pdf.

Murthy, R., Sedlar, E. (2007) "Flexible and efficient access control in oracle". In: ACM SIGMOD 2007, pp. 973-980, Beijing.

Needham, R. M., Schroeder, M.D. (1978) "Using Encryption for Authentication in Large Networks od Computers". Communications of the ACM 21(12) .

Prasanna, D., R. (2009) Dependency Injection. Manning Publications Co.

Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E. (1996) Role-based access control models. IEEE Computer, vol. 29, no. 2, pp 38-47.

TPCH (2008) "TPC Benchmark H Standard Specification Revision 2.8.0". *Transaction Processing Perfermance Council.* http://www.tpc.org/tpch/spec/tpch2.8.0.pdf.

Yang, L. (2009) "Teaching database security and auditing". ACM SIGCSE'09, v.1, issue 1, pp. 241—245.