UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

ADOÇÃO DE JBOSS DROOLS NO DESENVOLVIMENTO DE SISTEMAS

MATEUS ARTUR SCHNEIDERS

BLUMENAU
2011

2011/2-24

MATEUS ARTUR SCHNEIDERS

ADOÇÃO DE JBOSS DROOLS NO DESENVOLVIMENTO DE SISTEMAS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Sistemas de Informação— Bacharelado.

Prof. Mauro Marcelo Mattos, Doutor - Orientador

BLUMENAU
2011

2011/2-24

ADOÇÃO DE JBOSS DROOLS NO DESENVOLVIMENTO DE SISTEMAS

Por

MATEUS ARTUR SCHNEIDERS

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Mauro Marcelo Mattos, Doutor – Orientador, FURB
Membro:	Prof. Aurélio Faustino Hoppe, Mestre – FURB
Membro:	Prof. Jacques Robert Heckmann, Mestre – FURB

Blumenau, 7 de dezembro de 2011.

Dedico este trabalho aos meus pais, aos meus amigos e ao meu orientador.

AGRADECIMENTOS

Aos meus pais, Ademar e Paula, que sempre me apoiaram e incentivaram nessa conquista.

Ao meu orientador, Mauro, por ter acreditado na proposta e na minha capacidade de concluir este trabalho.

Aos meus amigos, pela ajuda e o apoio.

Ao colegiado do curso de Sistemas de Informação da FURB, pelo aprendizado adquirido durante a minha graduação.

O sucesso nasce do querer, da determinação e persistência em se chegar a um objetivo. Mesmo não atingindo o alvo, quem busca e vence obstáculos, no mínimo fará coisas admiráveis.

José de Alencar

RESUMO

Este trabalho apresenta uma avaliação do impacto da adoção da ferramenta JBoss Drools na modelagem de lógica de negócio. Para a realização desta análise, desenvolveu-se uma aplicação de gerenciamento de processos jurídicos modelando uma lógica de controle de acesso utilizando JBoss Drools. Em seguida foi introduzido um novo requisito não funcional, a possibilidade de conceder acessos com data de vencimento, para poder realizar uma avaliação das mudanças necessárias tanto na parte da aplicação quanto na base de conhecimento. Os resultados demonstram JBoss Drools como sendo uma ferramenta com um grande potencial na modelagem de lógica de negócio.

Palavras-chave: Lógica de negócio. JBoss Drools. Vraptor.

ABSTRACT

This work presents an evaluation about the use of JBoos Drools on the modeling of business logic. For making this analisys, it was developed an application for managing judicial proceedings where an access control logic was modeled using JBoss Drools. As a second step, a new non-functional requirement was introduced, where access could be granted with an expiration date. A new version of the application was developed for making the evaluation of the needed changes in the application and knowledge base. The results demonstrate that JBoss Drools turns out to be a high relevant tool to control business logic on applications.

Key-words: Business logic. JBoss Drools. Vraptor.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API - *Application Programming Interface*

DBC - Desenvolvimento Baseado em Componentes

DRL - *Drools Rule Language*

JAR - *Java Archive*

JSP - *Java Server Pages*

LHS - *Left Hand Side*

MER - Modelo Entidade Relacionamento

MVC - *Model View Controller*

POA - Programação Orientada a Aspectos

POO - Programação Orientada a Objetos

RF - Requisitos Funcionais

RHS - *Right Hand Side*

RNF - Requisitos Não Funcionais

TI - Tecnologia da Informação

URL - *Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO

O objetivo da Tecnologia da Informação (TI) é prover vantagem competitiva às organizações, automatizando tarefas manuais, tornando processos mais seguros e auditáveis, melhorando a comunicação com outras organizações e/ou clientes. Sistemas projetados para atender essas necessidades podem estar ligados intrinsecamente ao negócio da organização. No entanto, organizações dinâmicas respondem às mudanças do mercado. Desta forma, cabe a TI apresentar soluções à altura para manter a sincronia entre as regras de negócio e TI, o que pode significar estar à frente da concorrência ou não (LIU, 2011).

Segundo Liu (2011), existem softwares que ajudam a responder a esses tipos de problemas de forma flexível e dinâmica. Os chamados motores de regra avaliam informações de acordo com regras de negócio previamente definidas. A empresa JBoss oferece uma plataforma de lógica de negócio chamada JBoss Drools, que possibilita programar regras de negócio declarativamente e gerenciá-las de forma dinâmica (JBOSS, 2011a). O JBoss Drools é dividido em cinco sub-projetos:

a) *Drools Guvnor* - sistema de gerenciamento de regras que permite a organização, versionamento, verificação e edição de regras;
b) *Drools Expert* - motor de regras da plataforma que executa regras de negócio dado um conjunto de fatos;
c) *Drools Flow* - motor de processos da plataforma que possui uma forma de integração com as regras de negócio;
d) *Drools Fusion* - motor de processamento de eventos complexos, que é uma forma de regra de negócio que leva em conta os aspectos temporais e *streaming* de eventos;
e) *Drools Planner* - utilizado para a resolução de problemas usando heurísticas que retornam resultados considerados “o melhor possível” para problemas que não possuem uma solução algorítmica definitiva.

Conforme Strandberg (2005), os motores de regras são um ótimo meio de coletar lógicas de negócio complexas e trabalhar com um conjunto de dados maior do que um humano poderia processar de forma eficiente

A Drools é uma implementação de motor de regras poderosa e flexível. Com a sua utilização, as aplicações ganham potencial para serem mais manuteníveis e extensíveis do que as que não fazem uso de um motor de regras (OLIVIERI, 2006).

Neste contexto, pretende-se desenvolver uma aplicação de um domínio hipotético, neste caso, para o gerenciamento de processos jurídicos modelando as regras de negócio com JBoss Drools. Em seguida, um novo requisito não funcional seria incluído, gerando uma nova versão da aplicação e possibilitando a avaliação da facilidade com que as mudanças podem ser feitas nesse contexto.

1.1 OBJETIVOS DO TRABALHO

O objetivo do presente é avaliar o impacto da adoção da ferramenta JBoss Drools na modelagem de lógica de negócio. Como objetivo específico tem-se:

a) disponibilizar uma aplicação *web* hipotética para gerenciamento de processos jurídicos modelando as regras de negócio em JBoss Drools.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está disposto em quatro capítulos.

No primeiro capítulo apresentam-se a introdução, os objetivos e como estão dispostos os assuntos em relação a sua organização.

No segundo capítulo tem-se a fundamentação teórica, onde apresentam-se os conceitos sobre regras de negócio, motor de regras, JBoss Drools e trabalhos correlatos.

No terceiro capítulo, é apresentado todo o ciclo de desenvolvimento do sistema, incluindo detalhes sobre a especificação, requisitos funcionais e não funcionais e operacionalidades do sistema com suas principais telas.

No quarto capítulo tem-se a conclusão e extensões.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são abordados os principais assuntos e conceitos que auxiliam o melhor entendimento do trabalho e que serviram como base para o desenvolvimento do sistema.

2.1 REGRAS DE NEGÓCIO

Segundo Business Rules Group (2001) uma regra de negócio pode ser definida seguindo duas perspectivas:

a) perspectiva do negócio;
b) perspectiva de sistemas de informação (SI).

Segundo a perspectiva do negócio, regra de negócio é uma diretiva destinada a influenciar ou guiar o comportamento do negócio, como o suporte à política de negócio que é formulada em resposta a uma oportunidade.

Segundo a perspectiva dos sistemas de informação, uma regra de negócio é uma sentença que define ou restringe algum aspecto do negócio. Pretende-se garantir a estrutura do negócio ou controlar a influência do comportamento do mesmo.

Já de acordo com Date (2000), as regras de negócio são a solução para o problema da necessidade de se escrever código de forma procedural, podendo-se especificar sistemas apenas de forma declarativa. As regras de negócio, segundo ele, nos permitem automatizar o processamento de negócios.

2.2 MOTOR DE REGRAS

Segundo Tizzei (2007), um motor de regras é um software projetado para executar regras de forma otimizada. Ele simula a capacidade humana de chegar a uma decisão através de um raciocínio lógico. Um motor de inferência possui uma série de regras que são comparadas com fatos e cada uma delas pode ou não ter uma condição para ser executada.

Conforme Griffin e Lewis (2002, p. 1) um motor de regras, ou motor de inferência, é um dos elementos que compõe um sistema especialista, realizando a execução das regras de lógica de negócio que estão registradas em uma base de conhecimento. Esse tipo de sistema tem se tornado um meio popular para representar grandes conjuntos de dados para uma determinada área de conhecimento.

Um motor de regras pode processar regras baseando se em centenas de milhares de fatos repetidamente de forma rápida e confiável. Nos negócios, o relacionamento complexo de centenas de regras operando em dezenas de fatos concorrentes, podem influenciar no resultado de decisões estratégicas importantes (STRANDBERG, 2005).

2.3 JBOSS DROOLS

O motor de inferência Drools é um projeto de código-fonte aberto que está associado ao JBoss (JBOSS, 2011a). O Drools é baseado em linguagens declarativas, suas regras são descritas em blocos *if/then* e pode não haver dependências entre elas. É possível também estabelecer uma sequência na qual as regras serão executadas (TIZZEI, 2007).

Segundo Tizzei (2007), os elementos que descrevem o funcionamento do motor de inferência Drools são:

a) memória de produção - é o nome dado ao local onde são armazenadas as regras que serão executadas;
b) memória de trabalho - comporta-se como um banco de dados global de símbolos que representam os fatos;
c) agenda - mecanismo onde é possível determinar a ordem de execução das regras.

O Drools utiliza o algoritmo de encadeamento para frente (*forward chaining*) em seu motor de inferência. Desta forma, o processo de inferência começa a partir da inserção dos fatos iniciais na memória de trabalho, conforme pode ser visto na Figura 1 – item A. Em seguida é iniciado o processo de verificação das regras contidas na base de conhecimento, cada regra contendo uma condição e uma ação a ser executada (Figura 1 – item B). As regras podem, durante a execução, inserir novos fatos na memória de trabalho, possibilitando o acionamento de outras regras (Figura 1 – item C).

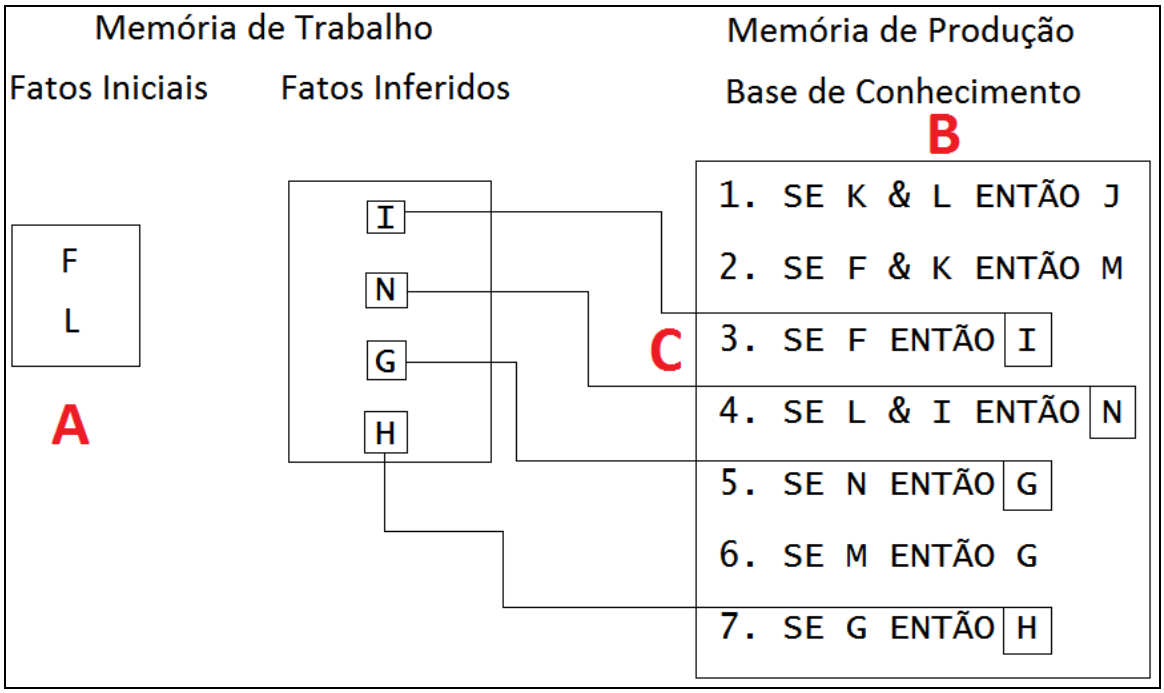
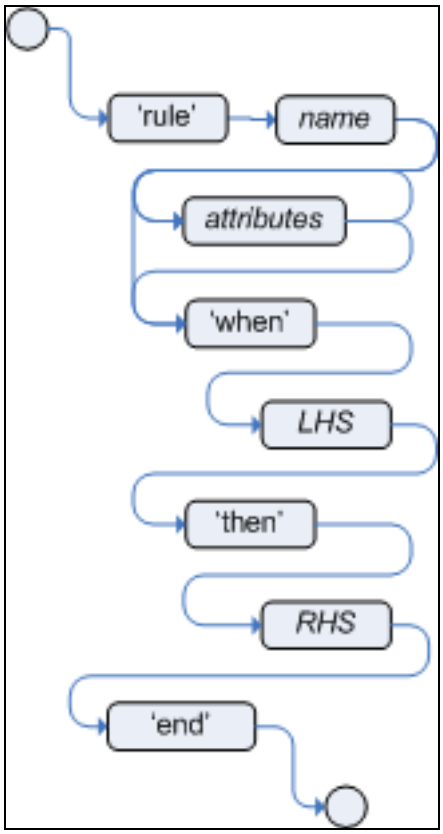


Figura 1 - Funcionamento do motor de regras

O motor de inferência Drools possui um arquivo que contém as regras e condições para executá-las. Este arquivo é processado pelo motor de inferência no qual ocorrem as comparações entre os fatos e as condições das regras. Além da linguagem própria do Drools, cuja extensão é *Drools Rule Language* (DRL), as regras podem ser implementadas nas linguagens Java, Groovy e Python. A linguagem usada para descrever os arquivos de regras do Drools é bastante amigável e permite que usuários com pouco conhecimento dela façam alterações de forma correta (TIZZEI, 2007).

Conforme Schmidt, Nascimento e Gorni (2010), o formato de regras do Drools baseia-se em um conceito simples, de um lado tem-se uma asserção, ou seja, as condições que ativam aquela regra em específico enquanto que do outro lado encontram-se as ações que deseja-se tomar no caso de essa regra ser ativada, sendo esses lados conhecidos por *Left Hand Side* (LHS) e *Right Hand Side* (RHS), respectivamente (Figura 2).



Fonte: JBOSS (2011c).

Figura 2 - Diagrama de Funcionamento do Drools

A Figura 2 apresenta a estrutura de definição de uma regra no Drools. São compostas por um nome identificador, alguns atributos opcionais, seção WHEN (quando) onde ficam as condições para a execução e seção THEN (então) onde ficam as ações a serem tomadas caso as condições forem verdadeiras.

2.3.1 Drools Guvnor

É um repositório centralizado de bases de conhecimento Drools com uma rica interface gráfica, editores e ferramentas que visam auxiliar no gerenciamento de um grande número de regras.

Em uma base de conhecimento Drools, também chamado de *package* (pacote) no Drools Guvnor, podem ser armazenadas regras, modelos, funções e processos, incluindo um sistema de versionamento para todos os recursos e controle de acesso. As definições das bases de conhecimento são mantidas pelo Drools como código fonte representado na Drools Rule Language (DRL), uma linguagem própria do Drools que contêm as definições para todos os recursos (JBOSS, 2011b).

No Drools Guvnor, bases de conhecimento são compostas primordialmente pela lógica de negócio de um domínio específico representada em forma de regras. Para representar a lógica através de regras, é preciso utilizar modelos para armazenar os dados que servem para contextualizar a área do conhecimento em foco. Esses modelos, também chamados de tipos de fatos, são entidades que representam uma unidade do negócio e que armazenam os dados (atributos) relevantes para a manutenção da lógica envolvida (SAUDATE, 2010). A Figura 3 apresenta um diagrama de atividades para explicar as etapas da integração do Drools, juntamente com imagens de utilização da ferramenta de própria autoria para auxiliar e complementar a fundamentação.

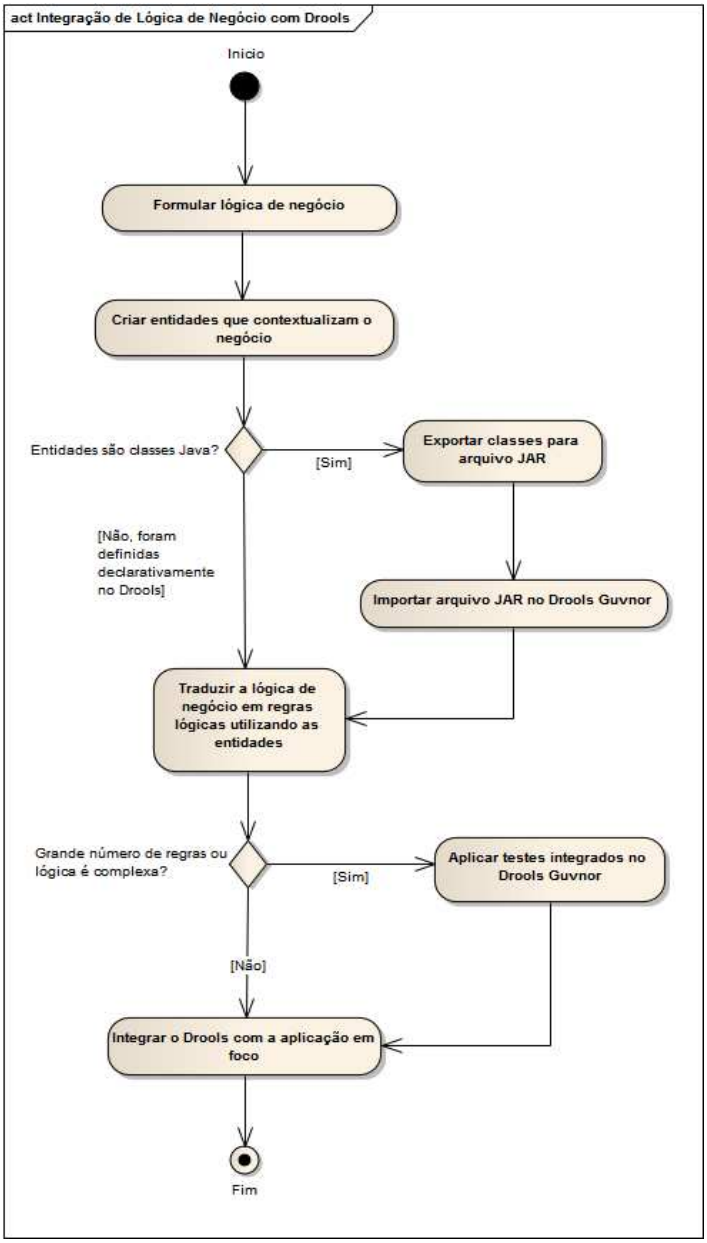


Figura 3 - Integração da lógica de negócio de uma aplicação com Drools

O primeiro passo é formular a lógica de negócio, ou seja, extrair as regras de negócio

de um domínio específico que serão integradas em uma aplicação. Uma dessas regras podendo ser, por exemplo, a negação de empréstimos para pessoas com menos de 21 anos em uma aplicação de avaliação da concessão de crédito.

Em seguida são criadas as entidades que contextualizam o negócio, ou seja, definição de entidades com dados que possibilitem a formação de regras lógicas. Essas entidades podem ser criadas de forma declarativa, ou por meio de importação de classes Java. A Figura 4 apresenta a definição dos tipos de fatos de forma declarativa referente à regra citada como exemplo anteriormente.

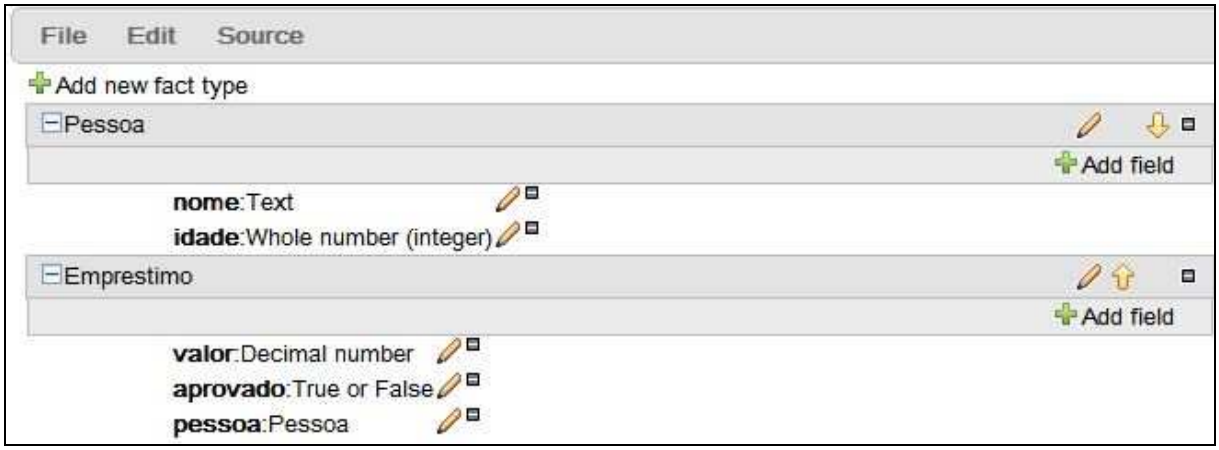


Figura 4 - Formulação de modelos declarativamente na interface do Guvnor

Uma alternativa à opção de definição de entidades declarativamente é a importação de classes. Após a implementação das entidades em forma de classes Java, é preciso exportar estas classes encapsulando-as em um arquivo do tipo Java Archive (JAR). A Figura 5 apresenta um exemplo de como fazer esta exportação pela ferramenta de desenvolvimento Eclipse.

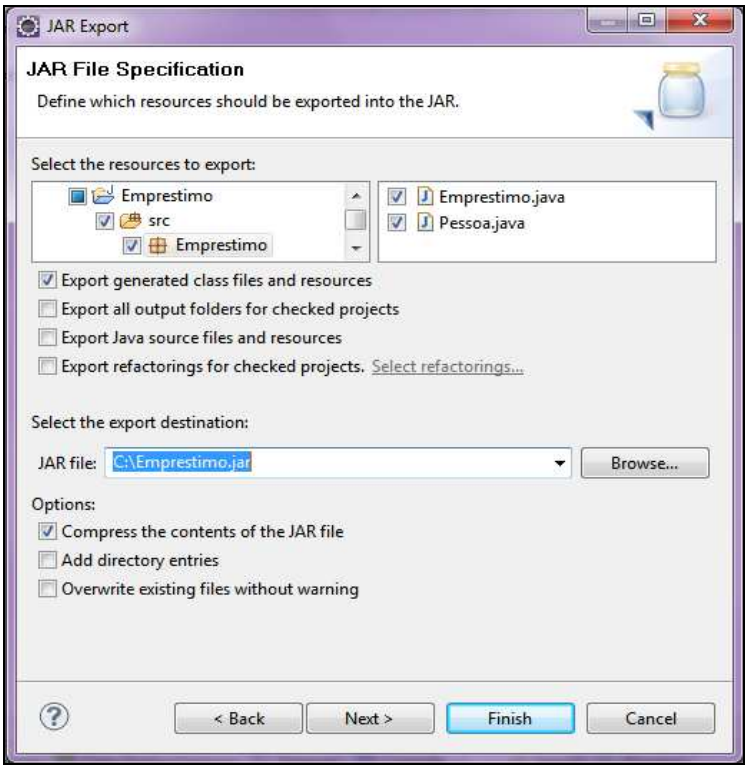


Figura 5 - Exportação de classes para arquivo JAR no Eclipse

Para importar arquivos JAR no Drools Guvnor, utiliza-se a opção “Upload POJO Model JAR” no menu “Create New”, são informados o nome e descrição do Modelo e em seguida, é feito o upload do arquivo JAR com as entidades (Figura 6).

Figura 6 - Upload de arquivo JAR para utilizar as classes como entidades

Após a criação das entidades, é preciso utilizá-las para traduzir a lógica de negócio em regras lógicas. Neste momento, é feita a criação das regras na base de conhecimento Drools utilizando a lógica do negócio já idealizada e as entidades. O Drools Guvnor fornece uma interface para a inserção e edição das regras que facilita a usabilidade e possibilita a manutenção de regras por pessoas que possuem menos conhecimento técnico (LUYPAERT, 2011). A Figura 7 apresenta um exemplo da regra para a negação de empréstimos citada anteriormente.

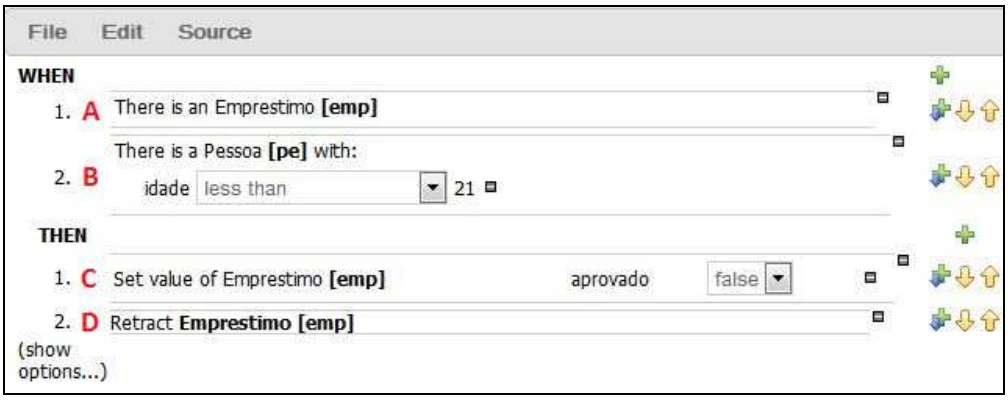
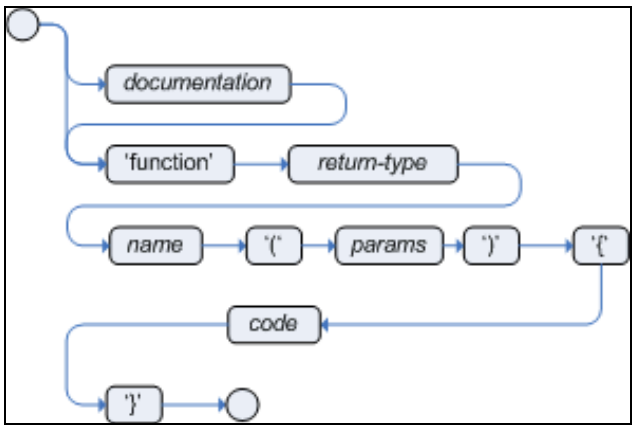


Figura 7 - Interface para criação e manutenção de Regras

Nesta regra, o LHS (condições para ativação da regra) é composto pela verificação da existência de um fato do tipo `Emprestimo` (Figura 7 - item A) e de outro do tipo `Pessoa` (Figura 7 - item B) com idade menor que 21. Se em um momento de inferência, o resultado destas condições for verdadeiro, o RHS da regra será executado, alterando-se o atributo `aprovado` do fato `Emprestimo` para “false” (Figura 7 - item C). É importante destacar o comando “retract” no RHS da regra (Figura 7 - item D), este possui o objetivo de fazer com que o objeto em questão (“emp”) seja desconsiderado pelo motor de regras a partir daquele momento, impedindo que esta mesma regra fosse disparada milhares de vezes consecutivamente.

O Drools Guvnor permite a criação de funções customizadas que podem ser utilizadas tanto em condições quanto em ações de regras incluídas na base de conhecimento. Funções são utilizadas como uma forma de representar uma determinada lógica que está envolvida no negócio. O principal benefício é o fato de conseguir manter a lógica de forma centralizada, podendo ser atualizada conforme as necessidades (JBOSS, 2011c). A Figura 8 apresenta um diagrama da composição de uma função.



Fonte: JBOSS (2011c).

Figura 8 - Composição da definição de uma Função no Drools

A definição de funções é muito similar à de um método declarado em uma classe Java,

a única diferença é a necessidade da palavra-chave '*function*' no início, no caso do Drools. Desta forma, define-se um tipo de retorno, parâmetros e o código em si (Quadro 1).

```
function Boolean temIdadeInvalida(Pessoa pessoa)
{
  return pessoa.getIdade() < 21;
}
```

Quadro 1 - Declaração de uma função

Para este caso a função tem o mesmo objetivo da regra de avaliação de empréstimo. A Figura 9 apresenta a utilização desta função no LHS de uma regra (Figura 9 - item A).

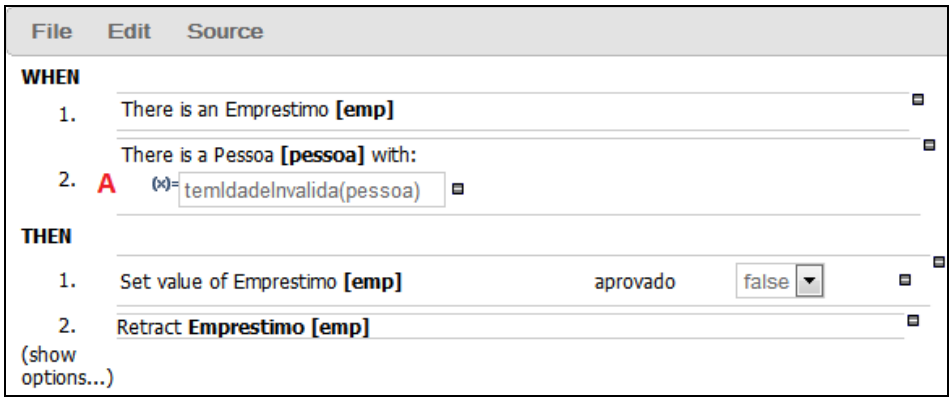


Figura 9 - Definição de regra utilizando função para verificação

Em aplicações com grande quantidade de regras e com lógicas complexas é imprescindível a utilização de testes unitários. O Drools Guvnor possui um recurso para realizar testes em regras criadas e armazenadas nas bases de conhecimento. É possível criar um ou mais cenários de testes para aferir condições normais e anormais de suas regras, como se fossem fluxos principais e fluxos alternativos de um caso de uso qualquer (FERREIRA, 2010). A Figura 10 apresenta um exemplo de cenário de testes que verifica a funcionalidade da regra definida na Figura 9.

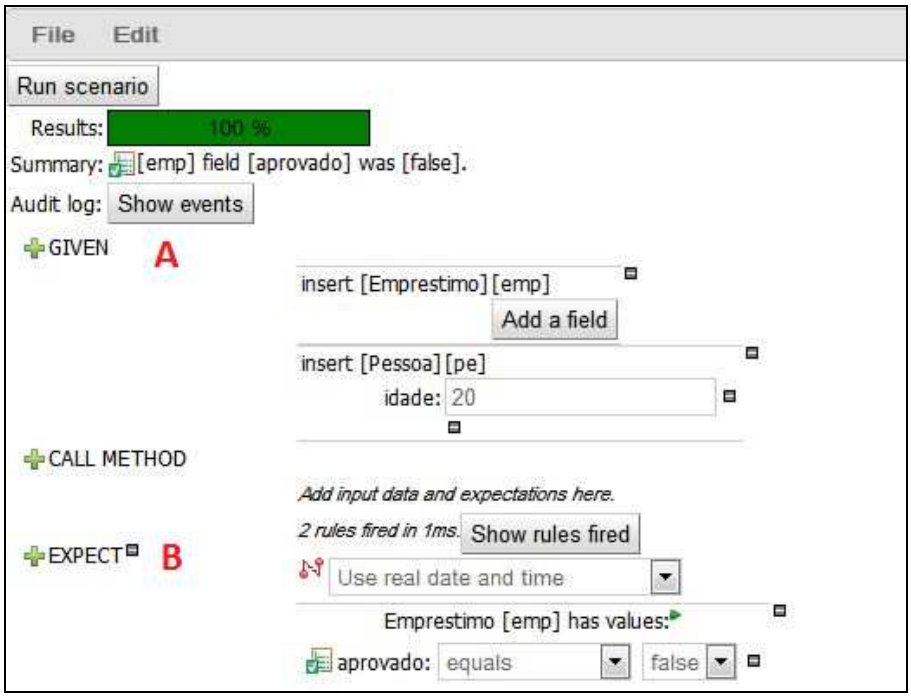


Figura 10 - Cenário de teste para verificação de regra

Na seção *GIVEN* (item A) da Figura 10, são determinados os dados de entrada, ou seja, os fatos necessários para contextualizar o cenário de teste. Na seção *EXPECT* (Figura 10 - item B) são definidas as verificações dos dados de saída seguindo as expectativas do cenário que está sendo testado. Após as ações de inferência do Drools, feita a partir dos dados fornecidos, é possível verificar se os objetivos foram atingidos e analisar a sequência de execução das regras para este cenário (FERREIRA, 2010).

A integração entre o motor de regras e bases de conhecimento com uma aplicação é feita a partir da utilização da *Application Programming Interface* (API) disponibilizada pelo Drools. A API do Drools permite a criação de sessões *stateful* (com estado) e *stateless* (sem estado) com métodos de inserção de fatos e execução das regras de uma base de conhecimento. A diferença entre os dois tipos de sessão é que na sessão *stateful* o estado dos fatos utilizados para disparar as regras é mantido, podendo ocorrer possíveis interferências no momento em que são inseridos novos fatos na sessão (KIJANOWSKI, 2008). As Figuras 11 e 12 apresentam exemplos de código para a criação e execução de regras com sessões *stateful* e *stateless* respectivamente.

```
1  //Cria agente com base em arquivo de propriedades
2  RuleAgent agent = RuleAgent.newRuleAgent("/arquivo.properties");
3  RuleBase rb = agent.getRuleBase();
4
5  //Cria sessão com estado
6  StatefulSession session = rb.newStatefulSession();
7
8  //Cria fatos
9  Emprestimo emprestimo = new Emprestimo();
10
11 Pessoa pessoa = new Pessoa();
12 pessoa.setIdade(21);
13
14 //Insere Fatos na sessão
15 session.insert(emprestimo);
16 session.insert(pessoa);
17
18 //Executa Regras
19 session.fireAllRules();
```

Figura 11 - Execução de regras com fatos em sessão com estado

```
1  //Cria agente com base em arquivo de propriedades
2  RuleAgent agent = RuleAgent.newRuleAgent("/arquivo.properties");
3  RuleBase rb = agent.getRuleBase();
4
5  //Cria sessão com estado
6  StatelessSession session = rb.newStatelessSession();
7
8  //Cria fatos
9  Emprestimo emprestimo = new Emprestimo();
10
11 Pessoa pessoa = new Pessoa();
12 pessoa.setIdade(21);
13
14 //Executa Regras
15 session.execute(new Object[]{ emprestimo, pessoa });
16
```

Figura 12 - Execução de regras com fatos em sessão sem estado

A execução das regras (linha 19 para *stateful* e linha 15 para *stateless*) é feita baseando-se em um arquivo de propriedades de configuração (linha 2). Este arquivo possui a *Uniform Resource Locator* (URL) para acessar o arquivo de definição das regras de uma base de conhecimento. Esta URL é fornecida pelo Drools Guvnor na hora da compilação de um pacote (Figura 13).

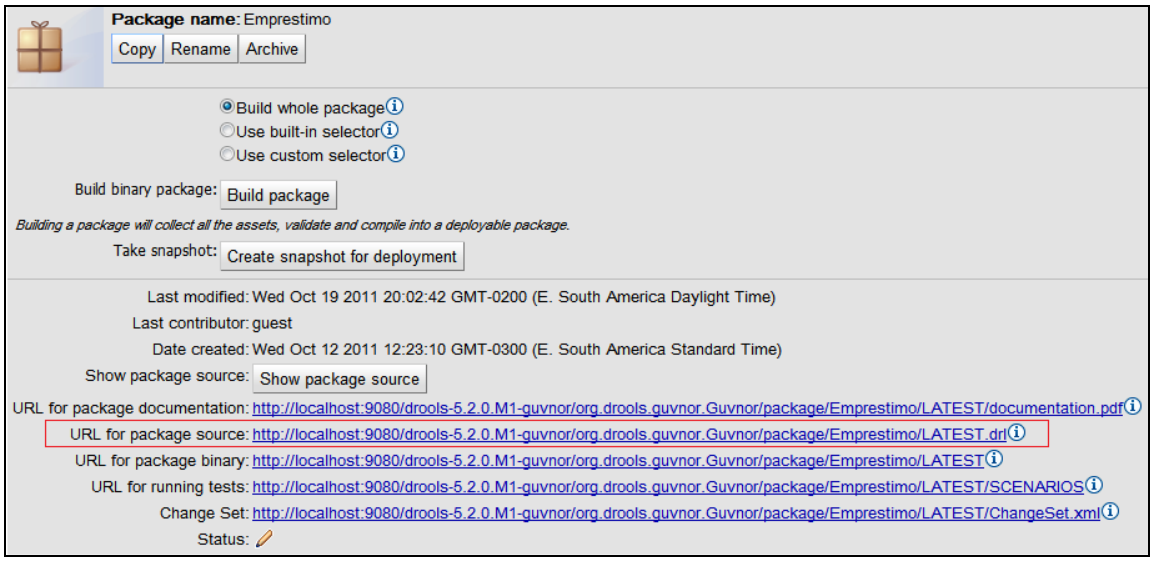


Figura 13 - Tela do Drools Guvnor onde é feita a compilação de pacotes

O Quadro 2 apresenta o conteúdo do arquivo de propriedades com a configuração de URL de acesso.

```
name = TCC
url = http://localhost:9080/drools/package/package_name/latest.drl
```
Quadro 2 - Conteúdo do arquivo de propriedades

2.4 TRABALHOS CORRELATOS

O trabalho de Silva (2005) apresentou uma análise comparativa entre Programação Orientada a Objetos (POO) e Programação Orientada a Aspectos (POA), onde aplicou-se os dois tipos de programação na implementação de requisitos que são de ampla influência no sistema, tal como controle de acesso às telas, gerando assim duas versões da aplicação. Foi avaliada a aplicabilidade da programação orientada a aspectos na implementação de requisitos ortogonais, que impactam de forma geral no sistema. A partir de uma primeira versão já desenvolvida, inclui-se um novo requisito onde pode-se avaliar a facilidade da implementação desta nova demanda com base na estrutura já existente. Assim como no presente trabalho, o requisito adicionado impactou fortemente na lógica antiga e pode-se validar, conforme os resultados apresentados, que a orientação aspecto é de fato uma tecnologia com grande potencial.

O trabalho realizado por Tizzei (2007) expos uma infra-estrutura de suporte à evolução em um repositório de componentes, para apoiar serviços de Desenvolvimento Baseado em Componentes (DBC), como a conversão de componentes para outros modelos de

implementação, por exemplo. A plataforma Drools foi utilizada como motor de inferência assim como para tornar as regras da aplicação mais modificáveis.

Schmidt, Nascimento e Gorni (2010) apresentaram, em seu trabalho de conclusão de curso um sistema especialista baseado em regras para auxiliar no ensino e avaliação dos diagnósticos e prescrições de enfermagem. A aplicação desenvolvida utilizou o Drools para armazenar as regras de um especialista na área de diagnósticos médicos. Desta forma, o motor de regras é chamado em uma funcionalidade especifica da aplicação onde é realizado o diagnóstico, tornando mais fácil a manutenção das regras a partir da plena definição dos fatos.

O Quadro 3 apresenta a relação das características presentes nos trabalhos relacionados. As características fazem referência ao modo com que foi implementado a lógica de negócio.

Características	**Silva (2005)**	**Tizzei (2007)**	**Schmidt, Nascimento e Gorni (2010)**
Lógica de negócio programada com orientação a objetos	X	X	X
Lógica de negócio programada com orientação a aspectos	X		
Lógica de negócio modelada com JBoss Drools		X	X

Quadro 3 - Características do trabalhos relacionados

Neste trabalho pretende-se implementar a lógica de negócio de um domínio hipotético (controle de acesso em uma aplicação de gerenciamento de processos jurídicos) utilizando o Drools para fornecer independência das regras em relação ao código fonte da aplicação. Em seguida pretende-se avaliar a facilidade proporcionada para realizar alterações na lógica existente a partir da inclusão de um novo requisito (concessão de acesso com data de vencimento).

3 DESENVOLVIMENTO

Neste capítulo são descritas as informações levantadas, detalhes da especificação, os diagramas de casos de uso e entidade relacionamento, a operacionalidade do sistema e ao final, os resultados e discussão.

3.1 LEVANTAMENTO DE INFORMAÇÕES

Por meio de conversas com o orientador, surgiu o interesse em aplicar a utilização de JBoss Drools em um estudo de caso para avaliar a sua eficiência no gerenciamento de regras de negócio. Desta forma, foi concebida a ideia de estar desenvolvendo uma aplicação em um domínio hipotético, neste caso, o gerenciamento de processos jurídicos. Para isso, decidiu-se aplicar o Drools através da implementação de dois cenários que obriguem uma adaptação na lógica de negócio da aplicação, podendo avaliar a facilidade com que as mudanças podem ser realizadas neste contexto.

3.2 ESPECIFICAÇÃO

Esta seção descreve os requisitos funcionais (RF) e não funcionais (RNF), bem como os diagramas de casos de uso e de entidade-relacionamento desenvolvidos para o sistema. A ferramenta Enterprise Architect (EA), em sua versão 7.5.845, foi utilizada na elaboração dos diagramas de casos de uso e a ferramenta case MySQL Workbench, versão 5.2.31 CE.

O Quadro 4 apresenta os requisitos funcionais e sua rastreabilidade com seus respectivos casos de uso.

Requisitos Funcionais	**Caso de Uso**
RF01: O sistema deverá permitir ao administrador o cadastro de advogados.	UC01
RF02: O sistema deverá permitir ao advogado o cadastro e manutenção de pessoas que poderão participar dos processos jurídicos.	UC03
RF03: O sistema deverá permitir ao advogado o cadastro e manutenção de tipos das participações de pessoas em processos jurídicos.	UC04

RF04: O sistema deverá permitir ao advogado o cadastro e manutenção de processos jurídicos.	UC05
RF05: O sistema deverá permitir ao advogado o cadastro e manutenção de participações de pessoas em processos jurídicos.	UC06
RF05: O sistema deverá permitir ao advogado manter os andamentos de processos jurídicos.	UC07
RF06: O sistema deverá permitir ao advogado o cadastro e manutenção de audiências de processos jurídicos.	UC08

Quadro 4 - Requisitos funcionais

O Quadro 5 apresenta os requisitos não funcionais do sistema.

Requisitos Não Funcionais
RNF01: O sistema deverá utilizar banco de dados MySQL 5 (ou superior)
RNF02: O sistema deverá ser desenvolvido na linguagem Java, utilizando Java Server Pages (JSP) e Framework Vraptor.
RNF03: O sistema deverá ser compatível com os navegadores Mozilla Firefox 3.6 (ou superior), Internet Explorer 8 (ou superior) e Chrome.
RNF03: O sistema deverá utilizar JBoss Drools para a lógica de negócio.

Quadro 5 - Requisitos não funcionais

O Quadro 6 lista as regras de negócio previstas para o sistema.

Regras de Negócio
RN01: Somente o administrador poderá cadastrar advogados.
RN02: Somente advogados poderão cadastrar e visualizar processos jurídicos, pessoas, tipos de participação, participações, andamentos e audiências.

Quadro 6 - Regras de Negócio

A Figura 14 apresenta o diagrama de casos de uso do estudo de caso. Os principais casos de uso estão descritos no Apêndice A.

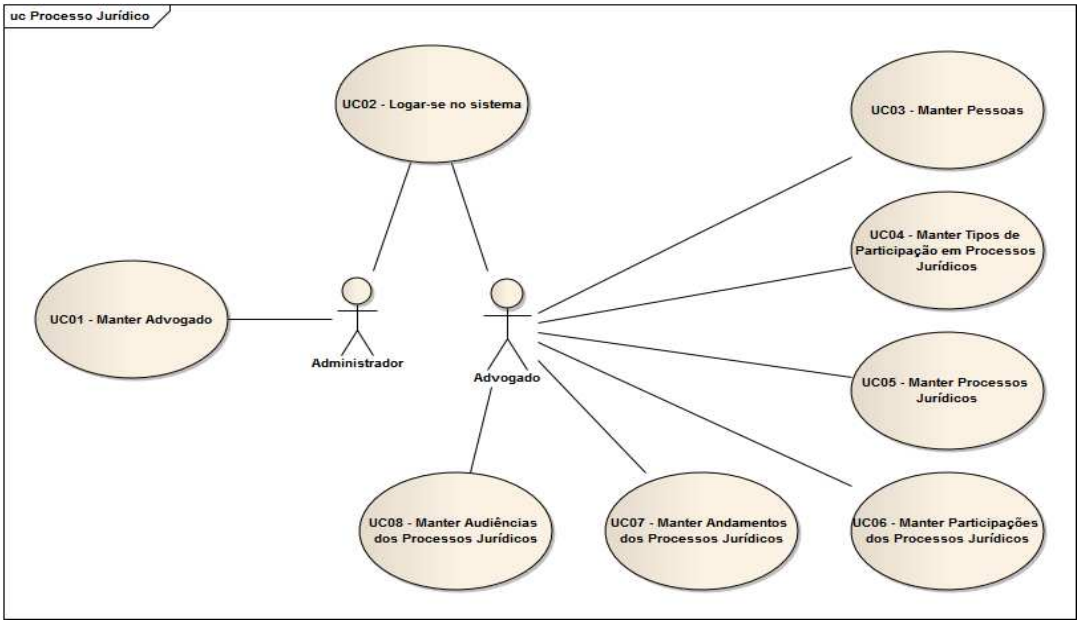


Figura 14 - Diagrama de casos de uso

A Figura 15 apresenta o modelo entidade e relacionamento que representam as entidades que serão persistidas no banco de dados. Para visualização dos atributos que compõem cada entidade, consultar o dicionário de dados no Apêndice B.

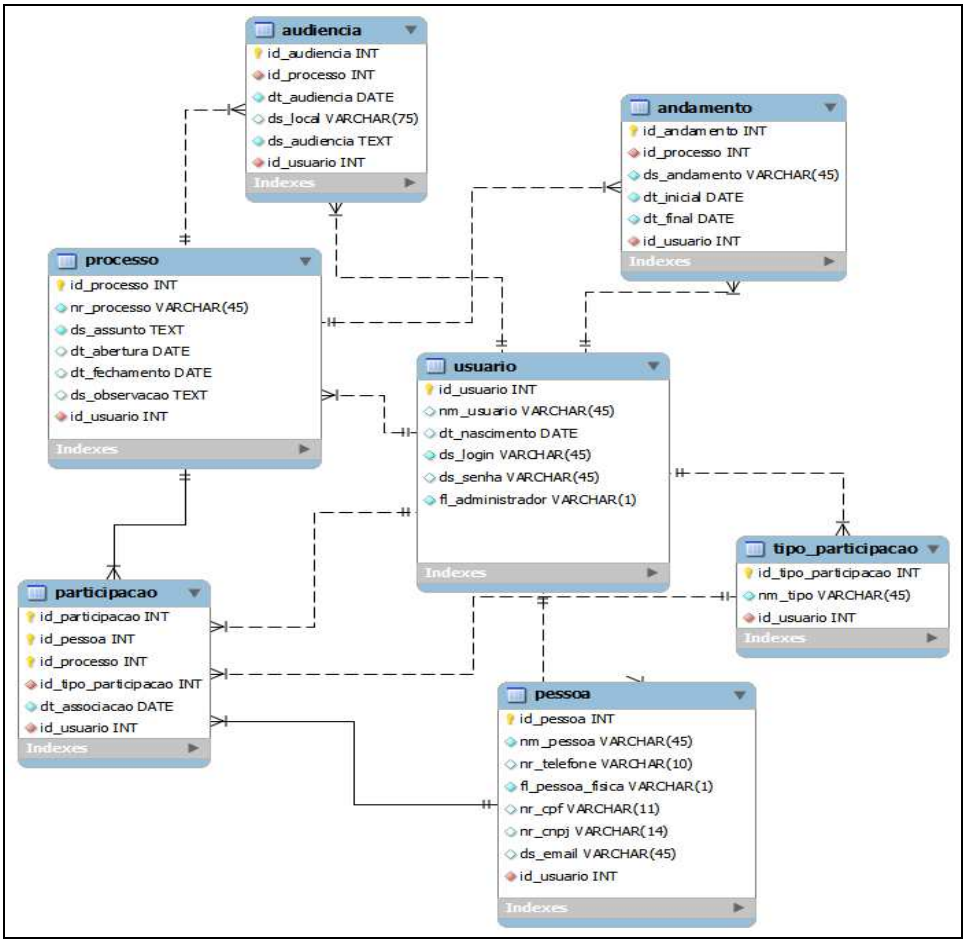


Figura 15 - Modelo Entidade Relacionamento (MER)

Segue abaixo uma breve descrição das entidades a serem utilizadas no desenvolvimento da aplicação:

a) usuario: entidade responsável por armazenar os usuários que utilizarão o sistema, sendo eles o administrador e os advogados por ele cadastrados;
b) pessoa: entidade responsável por armazenar as pessoas físicas e jurídicas que poderão ser associadas a determinados processos;
c) processo: entidade responsável por armazenar os processos jurídicos gerenciados por um advogado;
d) tipo_participacao: entidade responsável por armazenar os tipos de participação das pessoas que poderão ser associadas aos processos;
e) participacao: entidade responsável por armazenar as pessoas que de alguma forma participam de um processo jurídico;

f) andamento: entidade responsável por armazenar os andamentos que caracterizam os processos jurídicos;
g) audiencia: entidade responsável por armazenar as audiências realizadas de determinado processo.

3.3 IMPLEMENTAÇÃO

Nesta seção estão apresentadas informações sobre as ferramentas e técnicas utilizadas no desenvolvimento do sistema, juntamente com a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

No desenvolvimento do sistema, foram utilizados os softwares:

a) Apache Tomcat 7 – como servidor das aplicações *web*;
b) MySQL 5.5 – como banco de dados;
c) MySQL Workbench 5.2.31 CE – como gerenciador de banco de dados;
d) Java – como linguagem de programação;
e) *Java Server Pages* (JSP) – como tecnologia de servidor para desenvolvimento *web*;
f) Vraptor - como *framework Model View Controller* (MVC) Java para desenvolvimento *web*;
g) Hibernate – como *framework* para mapeamento objeto-relacional;
h) Eclipse – como ambiente de desenvolvimento;
i) Drools Guvnor - como sistema de gerenciamento de regras de negócio.

O Vraptor é um *framework* criado a partir de uma iniciativa brasileira, da Universidade de São Paulo. Utiliza o padrão de arquitetura MVC em Java focado no desenvolvimento rápido, simples e na fácil manutenção do código. Para maior produtividade, algumas boas práticas são aplicadas, como Convenção sobre Configuração, Injeção de Dependências e um modelo REST (CAVALCANTI, 2009).

3.3.2 Descrição da Implementação

A implementação foi dividida em 2 cenários, cada um representando uma versão da aplicação. O primeiro cenário foi desenvolvido para atender as regras de negócio definidas anteriormente no Quadro 6. No segundo cenário identificou-se um novo requisito funcional: possibilitar a concessão de acesso a uma ou mais telas com uma data de validade, garantindo que um advogado perca os acessos às telas definidas depois de uma determinada data (Regras com prazo de validade).

3.3.2.1 Cenário 1

Para o desenvolvimento do primeiro cenário, seguiu-se o fluxo de integração com o Drools apresentado na seção de fundamentação (seção 2.3.1 - Figura 3).

A partir da definição da lógica de negócio, foram criadas as entidades que possibilitam a criação das regras de negócio no Drools Guvnor. A Figura 16 apresenta o modelo conceitual das classes criadas.

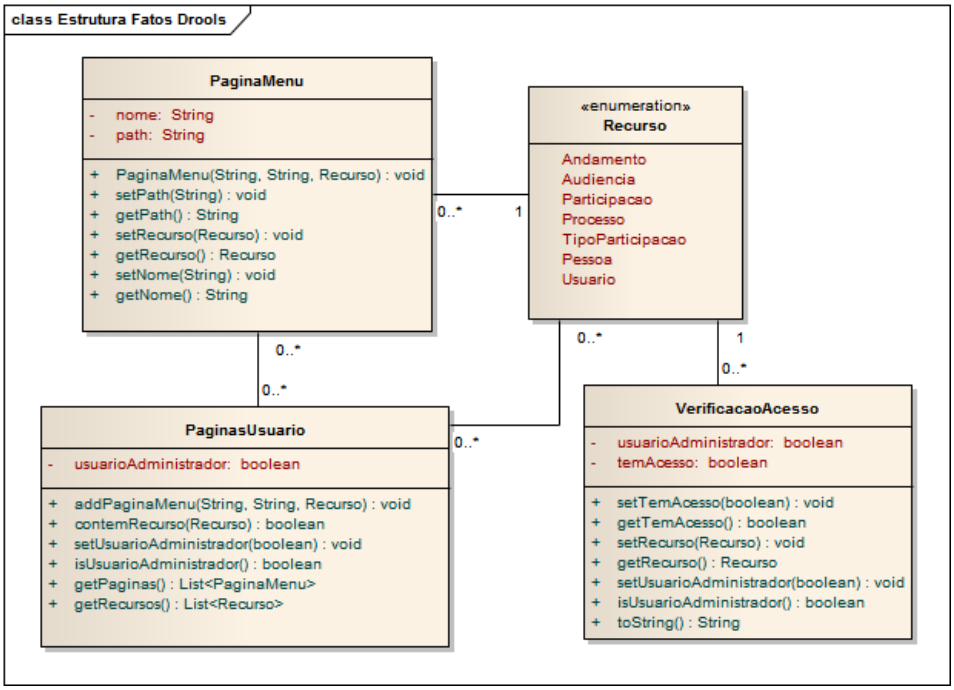


Figura 16 - Estruturas de Entidades para a criação das regras

A enumeração `Recurso` contêm a definição de todas as telas para as quais será possível controlar o acesso.

A classe `VerificacaoAcesso` é responsável por identificar se um usuário tem acesso a

determinado `Recurso`, e se a verificação é para um usuário administrador.

A classe `PaginaMenu` representa um item do menu da aplicação, portanto é composta por uma referência a um recurso, nome e o *path* (caminho) para o *link* da tela.

A classe `PaginasUsuario` representa a coleção de páginas que o usuário pode visualizar no menu, identificando se é usuário administrador.

No Drools Guvnor, foi criada uma base de conhecimento contendo as entidades definidas (Figura 16) e as regras para conceder acesso e definir os itens visíveis no menu da aplicação. A Figura 17 apresenta a definição da regra que concede acesso ao usuário administrador.

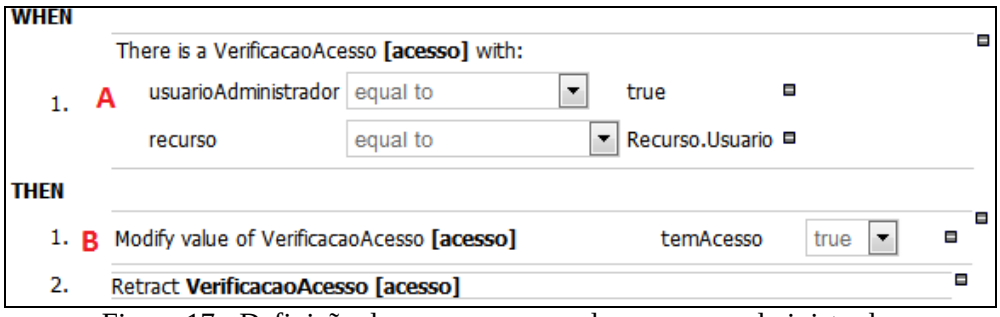


Figura 17 - Definição da regra que concede acesso ao administrador

A regra definida na Figura 17 baseia-se na existência de uma verificação de acesso para o recurso `Usuario` onde o usuário seja administrador (Figura 17 - item A). Desta forma concede o acesso ao cadastro de novos advogados (Figura 17 - item B).

A mesma verificação é feita para dar acesso ao advogado. A Figura 18 apresenta a definição da regra que concede acesso aos advogados.

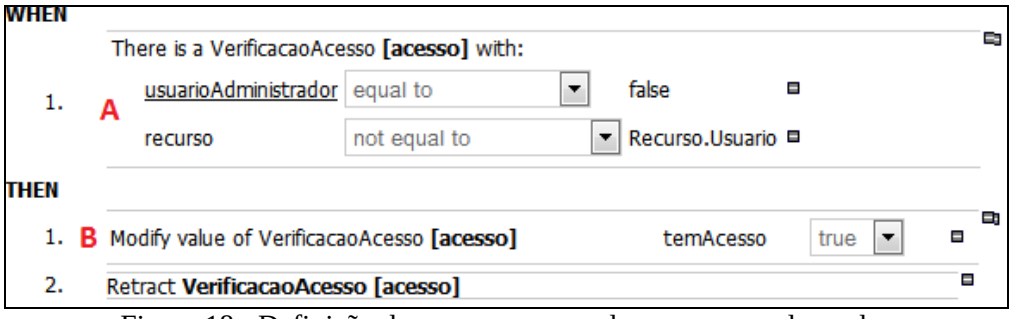


Figura 18 - Definição da regra que concede acesso aos advogados

A regra definida na Figura 18 baseia-se na existência de uma verificação de acesso a qualquer recurso diferente de `Usuario` onde o usuário não é administrador (Figura 18 - item A). Desta forma concede o acesso a todos os demais cadastros (Figura 18 - item B).

Outras duas regras foram criadas para a definição dos itens visíveis no menu dos advogados e do administrador. A Figura 19 apresenta a definição da regra para formação do menu do usuário administrador.

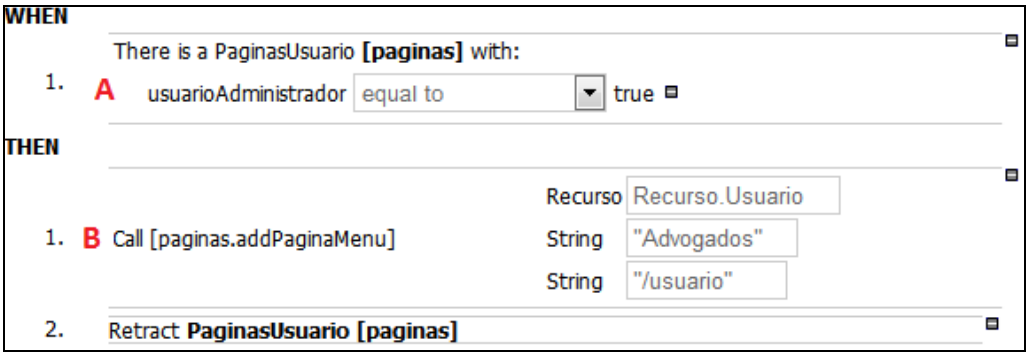


Figura 19 - Regra de definição dos itens do menu para o administrador

A regra definida na Figura 19 tem como condição a existência de um fato do tipo `PaginasUsuario` onde o usuário seja administrador (Figura 19 - item A). Se a condição for atendida, adiciona-se um novo item para o menu do usuário (Figura 19 - item B) chamando o método `addPaginaMenu` do objeto "paginas" passando os parâmetros referentes ao recurso, texto do item e *path* do *link*.

A mesma lógica é utilizada para os usuários advogados. A Figura 20 apresenta a definição da regra para formação do menu dos usuários advogados.

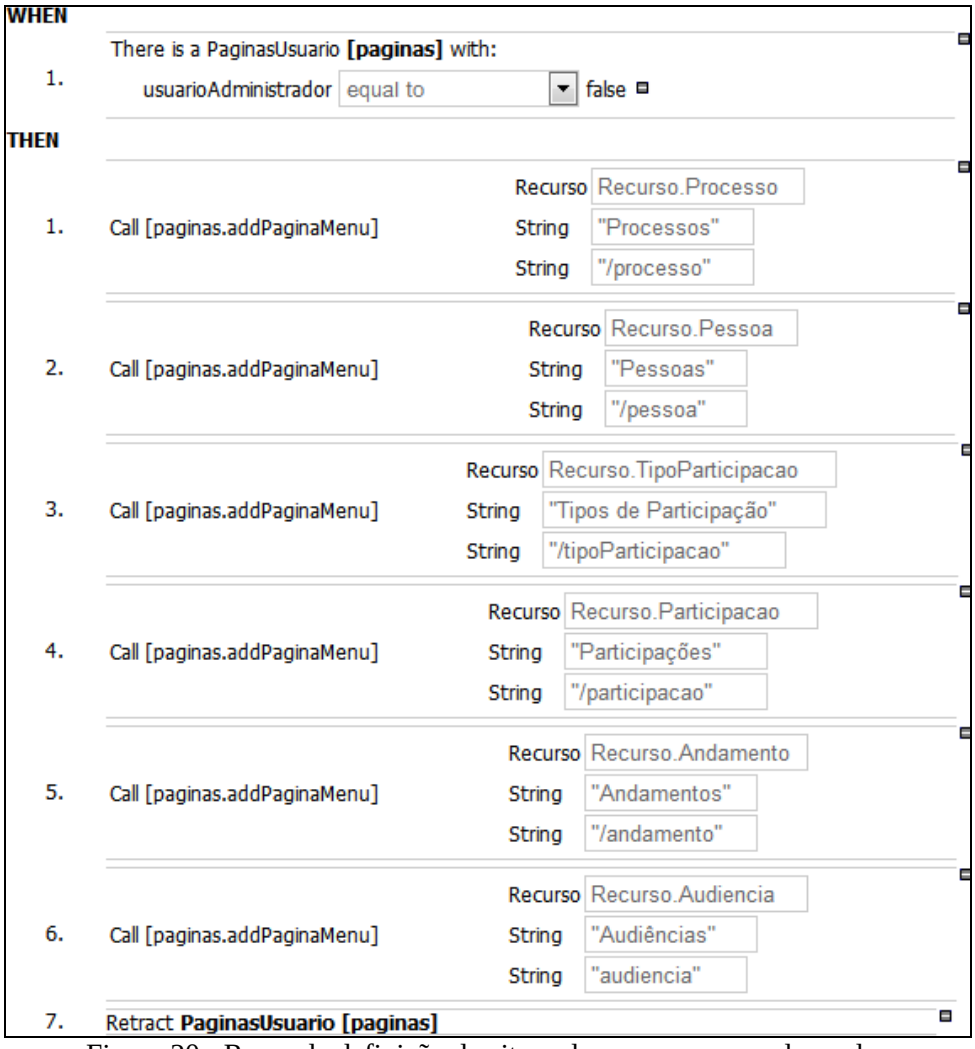


Figura 20 - Regra de definição dos itens do menu para os advogados

A estrutura da aplicação foi desenvolvida baseando-se em um contexto MVC fornecido pelo *framework* Vraptor. A Figura 21 apresenta o modelo conceitual da estrutura utilizada para implementar o gerenciamento das requisições e o controle de acesso através da integração com o Drools.

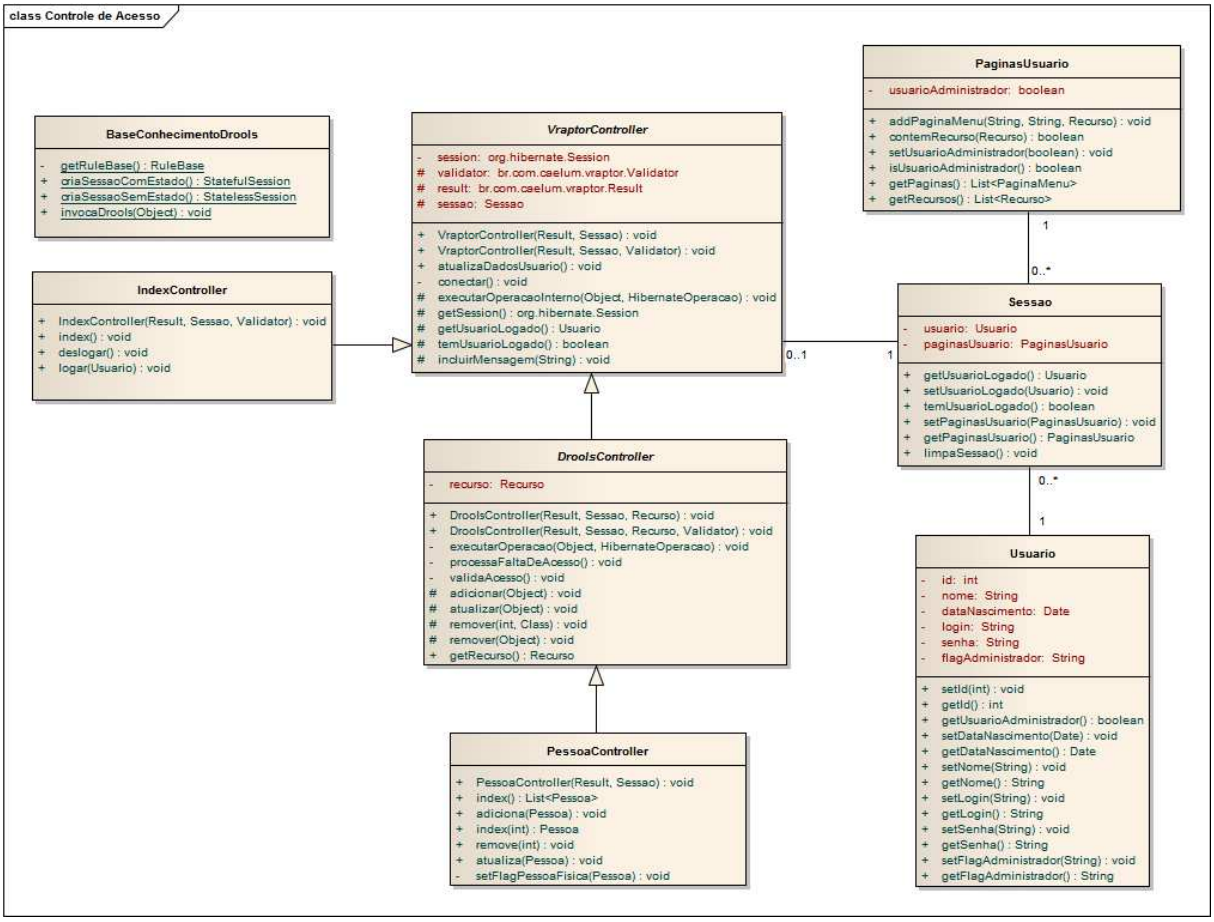


Figura 21 - Modelo conceitual da estrutura da aplicação

A classe `BaseConhecimentoDrools` é responsável por integrar a aplicação com o Drools. Fornece métodos estáticos que possibilitam a execução das regras a partir da inserção de fatos.

A classe `PaginasUsuario` representa a coleção de páginas que o usuário pode visualizar no menu, identificando se é usuário administrador.

A classe `Usuario` representa um individuo que esta cadastrado na aplicação, podendo ser como administrador ou simplesmente advogado.

A classe `Sessao` é a estrutura definida para armazenar os dados do usuário na sessão da aplicação. Possui uma referência a um objeto do tipo `Usuario` e a outro objeto do tipo `PaginasUsuario` onde armazena os itens do menu que este usuário pode visualizar. Essas informações são formadas no momento em que um usuário realiza o acesso à aplicação.

A classe abstrata `VraptorController` faz o gerenciamento de recursos do *framework*

Vraptor, tais como objetos para validação (`Validator`), alteração na resposta enviada como retorno da requisição (`Result`) e Hibernate para realizar o acesso aos dados.

A classe abstrata `DroolsController` herda de `VraptorController` e abstrai a validação de acesso a determinado recurso utilizando a integração com o Drools. O recurso é obrigatoriamente passado pelo construtor desta classe, ou seja, todas as classes controladoras que poderão ter o acesso bloqueado via Drools deverão herdar desta classe implementando um construtor que passe o respectivo recurso para a construção da super classe. A Figura 26 apresenta a definição da classe juntamente com os métodos de validação e de processamento em caso de falta de acesso.

```
1  public abstract class DroolsController extends VraptorController {
2
3      private Recurso recurso;
4
5      public DroolsController(Result result, Sessao sessao, Recurso recurso, Validator validator)
6      {
7          super(result, sessao, validator);
8          this.recurso = recurso;
9
10         try
11         {
12             validaAcesso();
13         }
14         catch(AcessoNegadoException ex)
15         {
16             processaFaltaDeAcesso();
17         }
18     }
19
20     private void validaAcesso() throws AcessoNegadoException
21     {
22         if (super.temUsuarioLogado())
23         {
24             Usuario usuarioLogado = super.getUsuarioLogado();
25
26             VerificacaoAcesso acesso = new VerificacaoAcesso();
27
28             acesso.setRecurso(this.recurso);
29             acesso.setUsuarioAdministrador(usuarioLogado.getUsuarioAdministrador());
30
31             BaseConhecimentoDrools.invocaDrools(acesso);
32
33             if(!acesso.getTemAcesso())
34             {
35                 throw new AcessoNegadoException();
36             }
37         }
38     }
39
40     private void processaFaltaDeAcesso()
41     {
42         ////Redireciona para pagina de falta de acesso
43         result.redirectTo(AcessoInvalidoController.class).index();
44     }
```

Figura 22 - Definição da classe `DroolsController`

Na linha 12 da Figura 26 (chamada ao método `validaAcesso`) é feita a verificação de acesso para o usuário referente ao recurso determinado. Se o usuário não tiver acesso ao recurso (linha 16), é feito o redirecionamento para uma página que informa o indivíduo sobre

a falta de acesso (Figura 23).

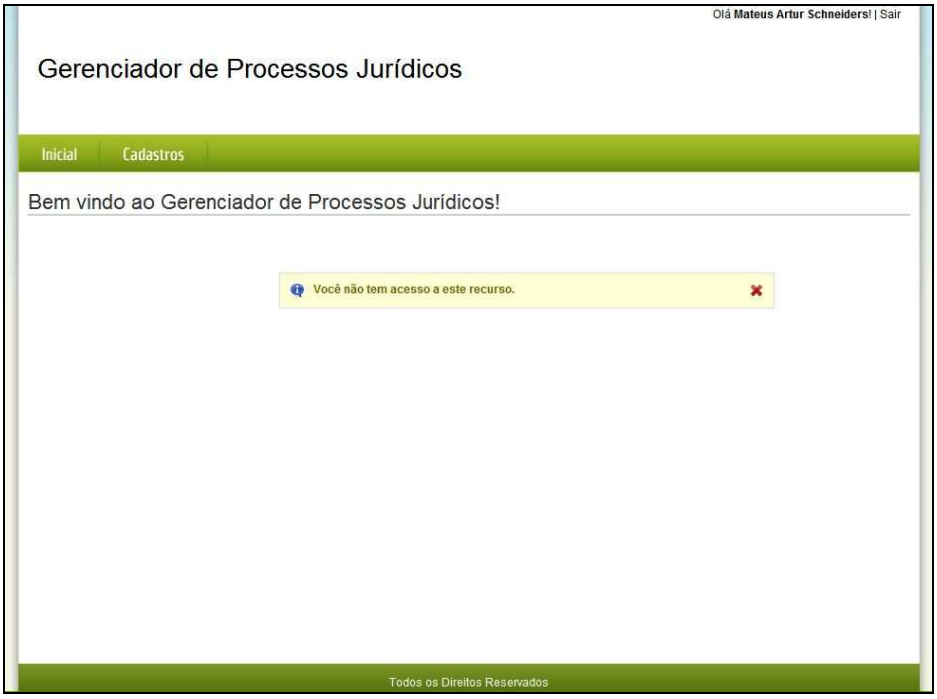


Figura 23 - Página de acesso inválido

A classe `IndexController` representa o gerenciador das requisições para a página inicial da aplicação, incluindo o *login* e as boas vindas ao usuário após a autenticação. Possui as funções básicas para realizar o acesso dos usuários já armazenando na sessão suas informações de acesso. O Figura 24 apresenta a definição do método de *login*.

```
1  public void logar(Usuario advogado)
2  {
3      List<Usuario> a = getSession().createCriteria(Usuario.class)
4                                    .add(Restrictions.eq("login", advogado.getLogin()))
5                                    .add(Restrictions.eq("senha", advogado.getSenha())).list();
6
7      if(a.size() > 0)
8      {
9          Usuario usuarioLogado = a.get(0);
10         PaginasUsuario pagsUsuario = new PaginasUsuario();
11
12         pagsUsuario.setUsuarioAdministrador(usuarioLogado.getUsuarioAdministrador());
13
14         //Chama motor de regras do Drools
15         BaseConhecimentoDrools.invocaDrools(pagsUsuario);
16
17         //Altera informações do usuário na sessão
18         super.sessao.setUsuarioLogado(usuarioLogado);
19         super.sessao.setPaginasUsuario(pagsUsuario);
20
21         super.atualizaDadosUsuario();
22
23         result.redirectTo(IndexController.class).index();
24     }else
25     {
26         validator.add(new ValidationMessage("Login ou Senha esta incorreto.", "Erro"));
27
28         //Indica para onde vai redirecionar
29         validator.onErrorUse(Results.page()).of(IndexController.class).index();
30     }
31 }
```

Figura 24 - Método de *login*

No método de *login* definido na Figura 24, primeiramente é feito a verificação das

credenciais do usuário (linha 3 e 7). Em seguida, caso o usuário esteja autenticado, é realizada a chamada ao motor de regras (linha 15) e a atualização dos dados na sessão do usuário (linha 18 e 19).

A classe `PessoaController` representa o gerenciador das requisições para a página de cadastro de pessoas. Herda da classe abstrata `DroolsController` para que o acesso seja verificado a cada requisição. As outras páginas também seguem este padrão, portanto, a página de pessoas está sendo citada meramente para exemplificar a estrutura utilizada.

3.3.2.2 Cenário 2

A segunda versão da aplicação foi desenvolvida para atender aos mesmos requisitos do primeiro cenário, possibilitando ainda a concessão de acesso para determinadas telas até uma certa data e hora.

Algumas alterações foram feitas na estrutura da aplicação para suportar o novo requisito (Figura 25).

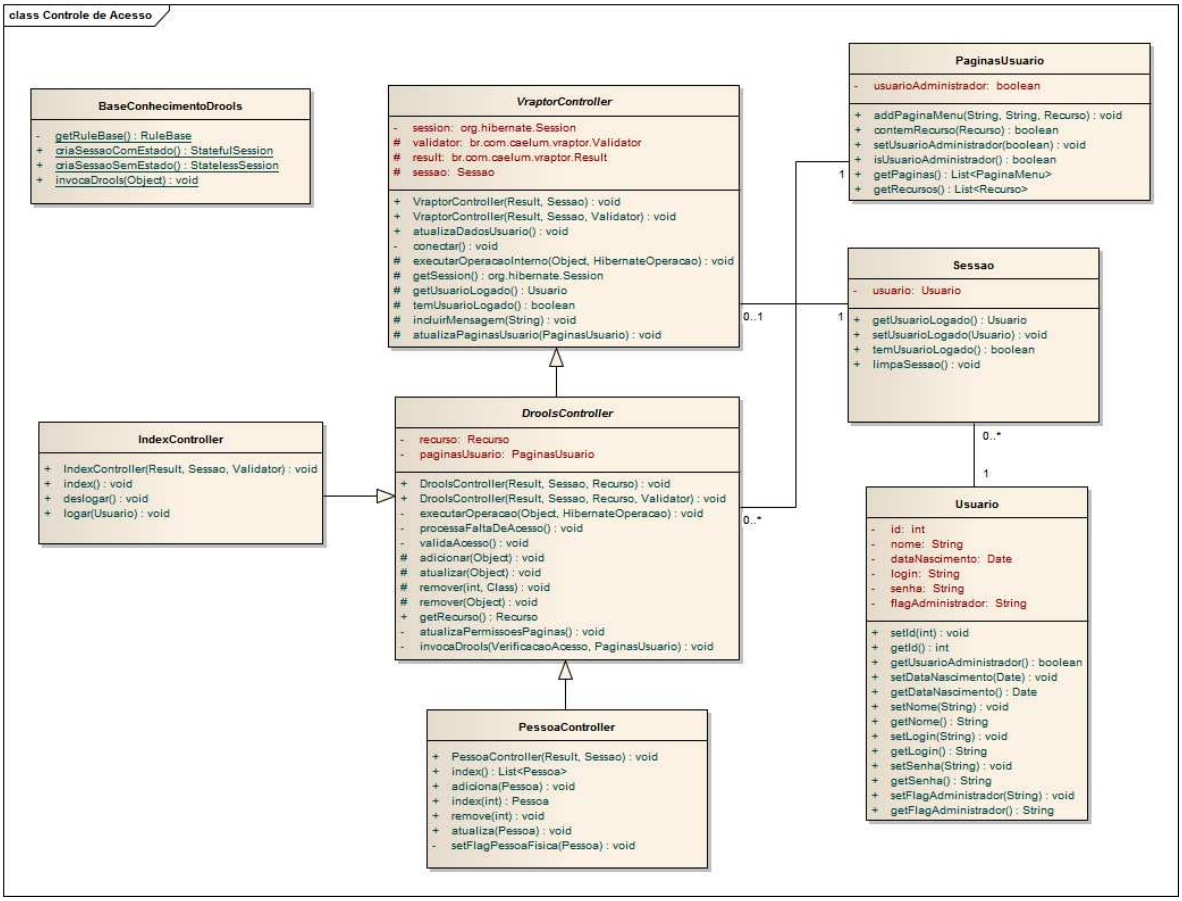


Figura 25 - Segunda versão do modelo conceitual da estrutura da aplicação

As mudanças para atender a nova regra foram necessárias pela forma com que a integração com o JBoss Drools foi realizada na primeira versão. No primeiro cenário, o Drools era chamado em dois momentos diferentes, no momento do *login* de um usuário (armazenando na sessão as páginas do menu de um usuário) e nas requisições feitas às páginas da aplicação (verificando se havia permissão para o acesso). Desta forma, não havia preocupação com mudanças nos acessos de um usuário durante a utilização da aplicação.

As seguintes alterações foram feitas na estrutura básica da aplicação:

a) `Sessao` – o objeto com as páginas do menu do usuário (`PaginasUsuario`) deixou de estar armazenado na sessão devido ao novo contexto de possíveis mudanças durante a utilização da aplicação;

b) `DroolsController` – passou a fazer o gerenciamento das páginas acessíveis no menu do usuário, fazendo um novo acesso ao Drools a cada nova requisição em qualquer tela da aplicação;

c) `IndexController` – deixou de depender dos dados que antes eram armazenados na sessão e passou a herdar de `DroolsController` para ter acesso à coleção de páginas do menu atualizadas.

Para este novo cenário foi criada uma nova base de conhecimento no Drools Guvnor para separar os componentes de ambas as versões. Primeiramente foi criada uma função para possibilitar uma condição adicional no LHS das regras podendo impedir que a mesma seja executada se a data atual for maior que a data determinada (Quadro 7).

```
function Boolean dataAtualMenorQue(String data)
{
  DateFormat formatter = new SimpleDateFormat(“dd/MM/yyyy H:m:s”);
  Date dtVencimento = (Date) formatter.parse(data);

  Calendar dataAtual = Calendar.getInstance();
  Calendar dataVencimento = Calendar.getInstance();

  dataVencimento.setTime(dtVencimento);
  return !dataVencimento.before(dataAtual);
}
```

Quadro 7 - Função para verificação de data

Utilizando esta função, pode-se garantir que o RHS de uma regra seja executado somente se a data parametrizada for maior que a atual. Desta forma é possível garantir que um usuário tenha acesso a algum recurso somente de forma temporária.

Para a exemplificação da funcionalidade e criação das regras, foi criado um cenário hipotético onde o advogado teria acesso temporário somente para a tela de cadastro de audiências.

Nesta nova base de conhecimento, as entidades utilizadas para a formulação das regras foram as mesmas do primeiro cenário, não necessitando de adaptações. Já a lógica de negócio foi reformulada para atender o novo contexto. Foram criadas as seguintes regras:

a) menu admin – concede acesso à tela de usuários para o administrador;
b) menu advogado – concede acesso às telas de cadastro aos advogados, deixando de fora a página de manutenção de audiências, a qual seria liberada de forma temporária para esses usuários;
c) menu advogado audiência – concede acesso temporário à tela de cadastro de audiências para usuários advogados;
d) acesso – concede acesso para um usuário à determinada tela com base na existência deste recurso na coleção de páginas do menu do fato `PaginasUsuario`. Retira os fatos da memória de trabalho, terminando a execução.

Para que o acesso seja concedido, é preciso garantir uma ordem na execução das regras, de tal forma que a ativação da regra “Acesso” poderá acontecer somente após as inclusões de páginas permanentes e temporárias do menu do usuário. Para garantir esta sequencia, as regras foram configuradas com as seguintes propriedades do Drools:

a) *salience* – valor numérico que indica a prioridade para a execução de uma regra, ou seja, quanto maior for o valor, maior a prioridade;
b) *no-loop* – evita que a mesma regra seja disparada duas vezes consecutivas. Se após a execução do RHS de uma regra as condições no LHS ainda estiverem sendo atendidas pelos fatos na memória de trabalho do Drools, esta regra não será disparada pela segunda vez consecutiva, mas poderá executar novamente depois da execução de outra regra qualquer.

A Figura 26 apresenta o fluxo de execução das regras pelo Drools. A partir da inclusão dos fatos e inicio da execução do motor de regras, foi realizada a projeção das regras de acesso do administrador (Figura 26 – item Menu Admin) e acesso temporário do advogado (Figura 26 – item Menu Advogado Audiência) para serem executadas por primeiro. Em seguida, é realizada a execução da regra de acessos permanentes do advogado (Figura 26 – item Menu Advogado). E por último, a regra que valida o acesso de forma genérica, verificando a existência do recurso entre os recursos adicionados nas regras anteriores (Figura 26 – item Acesso).

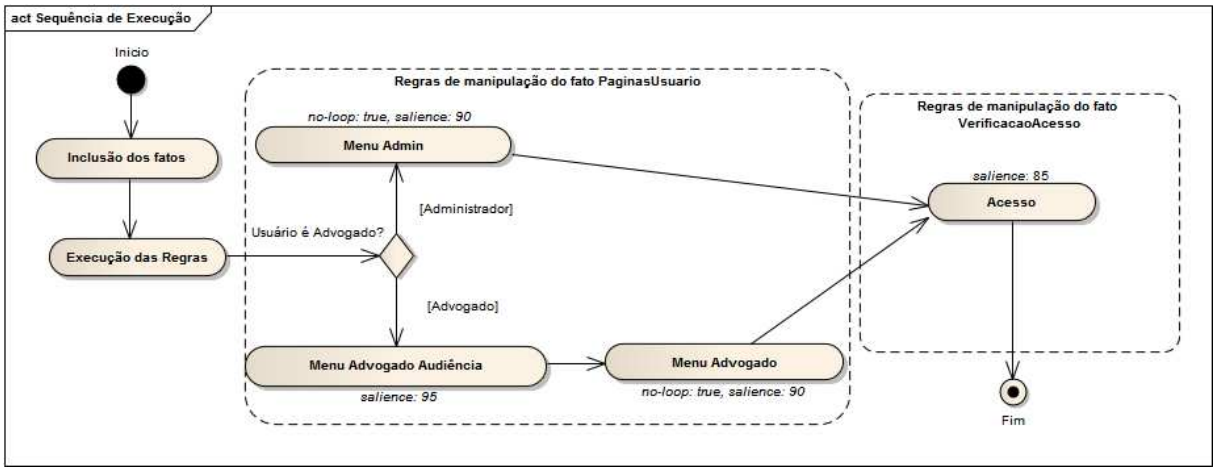


Figura 26 - Sequência de execução das regras no segundo cenário

A Figura 27 apresenta a definição da regra “Acesso”, que concede acesso a determinado recurso.

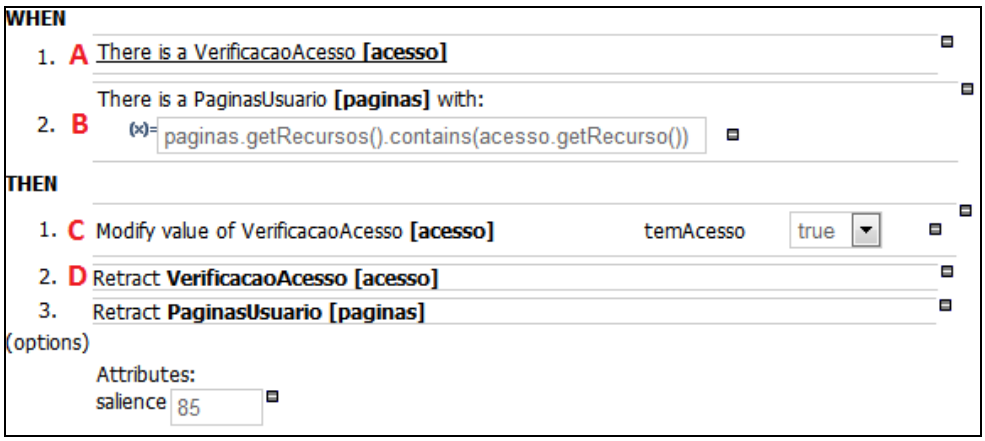


Figura 27 - Definição da regra "Acesso"

Na regra da Figura 27, verificou-se a existência de um fato do tipo `VerificacaoAcesso` (Figura 27 - item A) e outro do tipo `PaginasUsuario` (Figura 27 - item B), concedendo o acesso somente se o recurso estiver contido na coleção de páginas do menu, ou seja, se o recurso em questão estiver liberado para o usuário. Após a concessão de acesso (Figura 27 - item C), ambos os fatos são retirados da memória de trabalho do Drools a partir do comando “Retract” (Figura 27 - item D). Para garantir a execução desta regra somente depois do preenchimento dos acessos (representados por um fato do tipo `PaginasUsuario`), utilizou-se o atributo *salience* com um valor um pouco inferior ao colocado nas demais regras.

Para o usuário administrador, a primeira regra a executar é a “Menu Admin” (Figura 28).

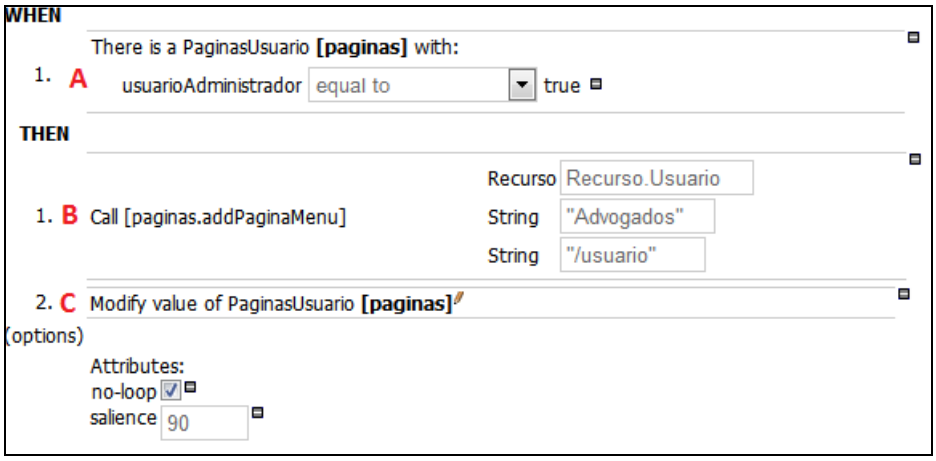


Figura 28 - Regra “Menu Admin”

A regra da Figura 28 valida se o usuário é administrador (Figura 28 - item A). Neste caso, libera o acesso à tela de usuários (Figura 28 - item B) e informa ao Drools que o fato `PaginasUsuario` foi alterado (Figura 28 - item C). Quando propriedades de um fato são alteradas no RHS de uma regra, o Drools automaticamente reavalia as possíveis regras que poderão executar a partir das novas condições. Essa reavaliação não é feita no caso de uma chamada de método em um fato qualquer, ou seja, ele não identifica que determinado fato foi alterado, até porque ele não sabe o que o método chamado faz. O atributo *no-loop* foi utilizado para evitar a execução consecutiva, pois o LHS continua sendo atendido após o disparo da regra. Através do atributo *salience* definiu-se a prioridade garantindo que esta regra execute antes da regra “Acesso”, que concede os acessos com base no fato `PaginasUsuario`.

Para os advogados, a primeira regra a executar é a “Menu Advogado Audiência”, onde concede-se o acesso temporário para à tela de audiências (Figura 29).

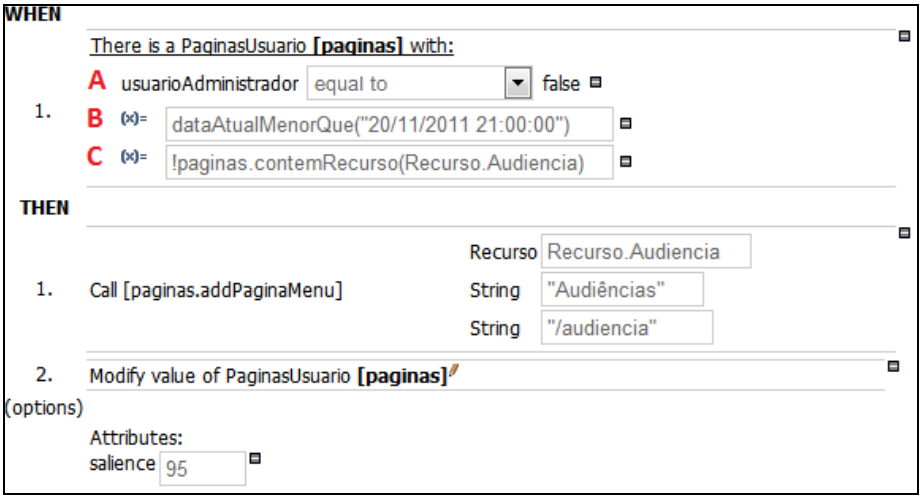


Figura 29 - Definição da Regra "Menu Advogado Audiência"

Na regra definida na Figura 29, foi utilizada a função criada anteriormente como condição para o disparo da regra (Figura 29 - item B), desta forma, garantindo que o usuário

poderá acessar a página de audiências somente até a data e hora estipuladas na chamada. Outra condição foi incluída, para garantir que a regra não execute novamente após o disparo da regra “Menu Advogado”, que concede os acessos permanentes para o usuário. Portanto, para o disparo da regra, verificou-se a inexistência do recurso `Audiencia` no fato `PaginasUsuario` (Figura 29 - item C), destacando ainda que o atributo *no-loop* não atenderia neste caso. Para garantir a prioridade de execução em frente as outras regras, utilizou-se o atributo *salience* com valor um pouco superior.

A próxima regra a ser disparada, realiza a concessão dos acessos permanentes aos advogados (Figura 30).

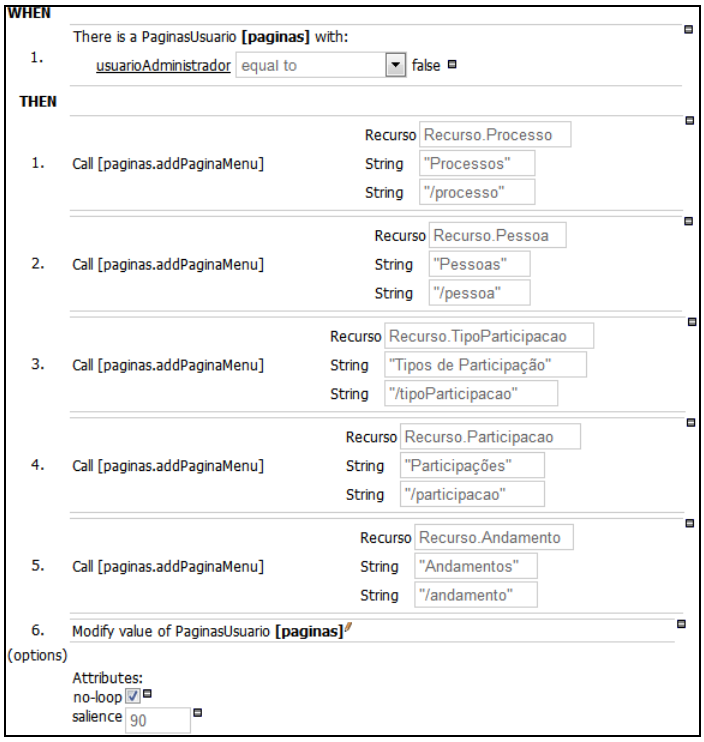


Figura 30 - Definição da regra "Menu Advogado"

A regra definida na Figura 30 concede os acessos permanentes aos advogados. Utiliza os atributos *no-loop* e *salience* para garantir uma execução controlada da regra, desta forma, assegurando que ela seja disparada após a regra “Menu Advogado Audiencia” e antes da regra “Acesso”.

3.3.3 Operacionalidade da Implementação

Esta subseção apresenta as principais telas da aplicação desenvolvida com uma breve apresentação de suas funcionalidades.

Ao acessar a aplicação *web*, a primeira tela é a de *login*. Nela serão informados o usuário e senha para realizar o acesso (Figura 31).

Figura 31 - Tela de acesso à aplicação

A navegação da aplicação é simples, contendo apenas um menu de referencia aos cadastros e um para ir à tela inicial. Os itens do menu variam de acordo com o tipo de usuário. O administrador é responsável por cadastrar novos advogados, enquanto que os advogados podem manter as informações relacionadas aos processos jurídicos (Figura 32).

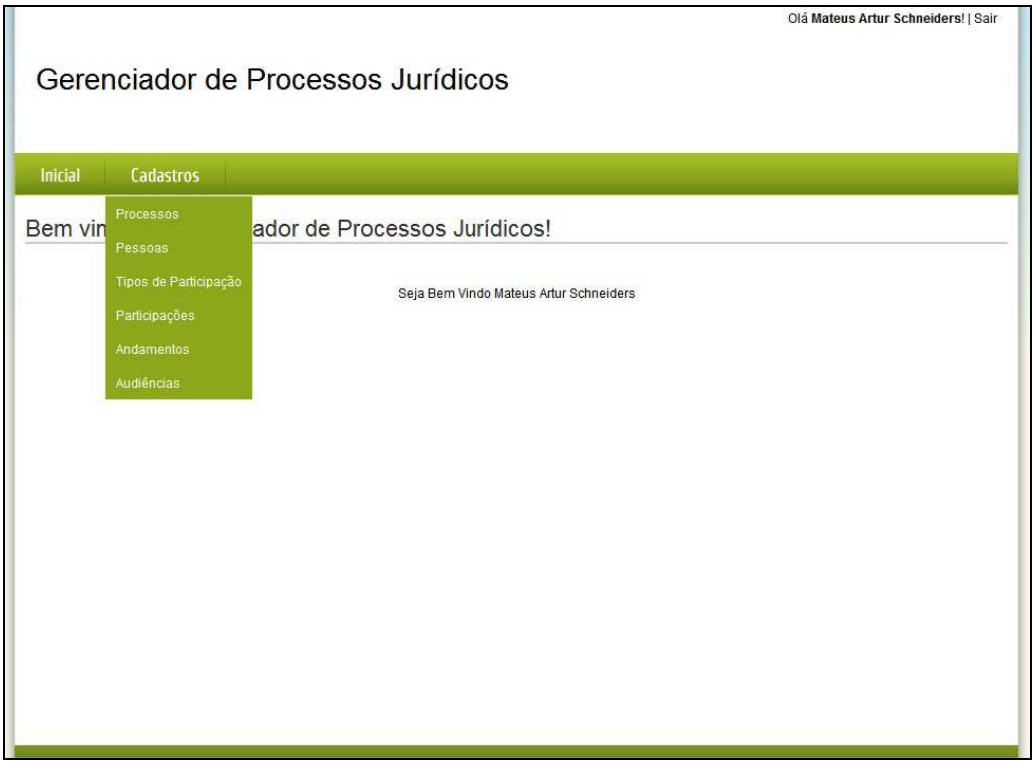


Figura 32 - Menu da aplicação para usuário advogado

A Figura 33 apresenta a tela para o cadastro de novos advogados, tarefa realizada pelo administrador da aplicação ao acessar o item “Advogados” no menu “Cadastros”.

Figura 33 - Cadastro de advogados

Os advogados tem a possibilidade de realizar vários cadastros. São telas padronizadas que apresentam o formulário de preenchimento para inserção de dados e a listagem dos dados

já existentes logo abaixo. O usuário tem a opção de editar ou excluir um registro conforme os ícones na coluna “Ações” na lista de itens cadastrados.

O primeiro cadastro é o de pessoas (Figura 34). São indivíduos que de alguma forma participam no andamento dos processos jurídicos.

Figura 34 - Cadastro de Pessoas

Os tipos de participação também são dados cadastrados pelo advogado (Figura 35). Nesta tela ele define quais as formas que as pessoas por ele cadastradas poderão participar em processo jurídicos.

Figura 35 - Cadastro de Tipos de Participação

O cadastro dos processos pode ser feito pelo advogado através do item “Processo” no menu “Cadastros” (Figura 36).

Figura 36 - Cadastro de Processos

Tendo um processo cadastrado, o advogado pode realizar as associações entre as pessoas e os processos através da tela de cadastro de participações (Figura 37).

Figura 37 - Cadastro de Participações

Para cada um dos processos é possível manter as respectivas audiências (Figura 38). Para a descrição desses eventos, o advogado pode informar a data, a descrição, ou seja, qual o

objetivo almejado e o local de realização da audiência.

Figura 38 - Cadastro de Audiências

Existe ainda uma extensão dos processos através do cadastro de andamentos (Figura 39). Desta forma o advogado pode associar determinados períodos de tempo á uma certa atividade realizada no processo jurídico em questão.

Figura 39 - Cadastro de Andamentos

3.4 RESULTADOS E DISCUSSÃO

Como se pode observar, o estudo de caso foi realizado partindo-se de um sistema de gerenciamento de processos jurídicos hipotético, tendo em vista que o foco principal foi a análise do impacto da utilização de JBoss Drools na gerência de regras de negócio de uma aplicação.

Em um primeiro momento foi desenvolvida uma versão da aplicação de gerenciamento de processos jurídicos aplicando a modelagem da lógica de negócio no ambiente oferecido pelo JBoss Drools (Guvnor). Neste cenário, os dados do usuário, dependentes das regras registradas na base de conhecimento, foram armazenados na sessão para persisti-los durante todo o acesso do usuário.

No segundo cenário, incluiu-se um novo requisito não funcional, onde seria possível definir uma data de vencimento (data e hora) para determinados acessos do usuário. Para a implementação deste novo contexto, foi preciso realizar algumas alterações na estrutura da aplicação. Deste modo, algumas informações que antes eram armazenadas na sessão precisaram ser migradas para conceder mais precisão, sendo que um usuário poderia perder o

acesso à alguma tela durante a utilização da aplicação. Para suportar esta nova demanda, a chamada ao motor de regras passou a ser feita a cada nova requisição, mantendo assim as informações de acesso sempre atualizadas.

Além das mudanças na aplicação, foram necessárias algumas alterações na base de conhecimento. A lógica das regras foi remodelada e uma nova função foi criada para possibilitar a verificação em relação à data de vencimento.

Diante dos dois cenários desenvolvidos e das mudanças realizadas para atender uma nova demanda, percebeu-se primeiramente a facilidade da utilização do Drools Guvnor na importação, declaração de entidades e na criação das regras lógicas associadas às bases de conhecimento. Em seguida é preciso apontar as alterações que foram necessárias na aplicação, onde precisou-se adaptar a própria chamada ao motor de regras, que pode ser considerado um aspecto de dependência entre o código da aplicação e as regras criadas nas bases de conhecimento. É importante destacar também um incremento na complexidade, em relação à primeira versão, principalmente no relacionamento dos fatos entre as regras.

O Quadro 8 apresenta as características dos trabalhos relacionados e do presente trabalho.

Características	**Este trabalho**	**Silva (2005)**	**Schmidt, Nascimento e Gorni (2010)**	**Tizzei (2007)**
Lógica de negócio programada com orientação a objetos	X	X	X	X
Lógica de negócio programada com orientação a aspectos		X		
Lógica de negócio modelada com JBoss Drools	X		X	X

Quadro 8 - Relação de características

Neste trabalho, as regras modeladas no JBoss Drools afetaram a aplicação em um contexto mais abrangente, sendo que trata-se de um controle de acesso a usuários e não de uma lógica de uma certa área de conhecimento, como foi apontado no trabalho de Schmidt, Nascimento e Gorni (2010). Desta forma, a integração do JBoss Drools com a aplicação é mais complexa, pois precisa permanecer em um ponto chave que atenda genericamente todas as telas da aplicação. Qualquer alteração neste quadro pode estar impactando diretamente na facilidade de futuras manutenções.

4 CONCLUSÕES

Os resultados obtidos apontam a importância da busca por novas opções na adequação dos softwares às mudanças do mercado, como sistemas de gerenciamento de regras de negócio. A evolução dos sistemas atuais apresenta cada vez mais novas demandas, introduzindo mais complexidade à lógica de negócio e justificando a busca por novas alternativas no desenvolvimento de software com maior poder de adaptação.

Sistemas gerenciadores de regras de negócio ajudam na construção e manutenção da lógica do negócio, centralizando tudo em uma base de conhecimento. No entanto, são estritamente dependentes da estrutura de entidades construída para a formulação da lógica. Em aplicações que possuem um grande número de regras e que se encontram em contextos de mudanças constantes é preciso de um controle maior para diminuir ao máximo o impacto das mudanças. Para isso, é necessário que os fatos envolvidos sejam facilmente identificáveis e que haja um planejamento adequado para a construção da estrutura conceitual dos fatos para minimizar o impacto de futuras adaptações.

A partir do experimento pode-se generalizar que a adoção do JBoss Drools oferece grande potencial em projetos com regras em constante mudança ou com grande quantidade, pois simplifica o processo de manutenção/adequação de regras de negócio, agilizando a liberação de uma nova versão da aplicação. Um dos aspectos que cabe a destacar é que a característica da aplicação escolhida limitou a aplicabilidade da ferramenta Drools, não exigindo um número de regras significativas.

É inevitável, se tratando de novas tecnologias, a necessidade de uma curva de aprendizado para sua plena utilização. No caso do JBoss Drools, especificamente, é necessária uma análise aprofundada dos recursos disponíveis e das reais necessidades da aplicação em foco, tendo em vista a tendência a uma maior complexidade de gerenciamento a medida em que a quantidade de regras aumenta. Desta forma, é preciso medir se o custo para este período de adaptação realmente vale a pena, ou seja, se pode refletir em benefícios futuramente.

4.1 EXTENSÕES

Para trabalhos futuros na área de integração e gerenciamento de regras de negócio,

sugere-se:

a) aprimorar utilização do recurso de saliência para permitir sintonia fina no comportamento do sistema;
b) selecionar um domínio de aplicação com maior número de regras de negócio e requisitos não funcionais.

REFERÊNCIAS BIBLIOGRÁFICAS

BUSINESS RULES GROUP. **Defining Business Rules - What Are They Really.** [S.l.], 2001. Disponível em: <http://www.businessrulesgroup.org/first_paper/br01c0.htm>. Acesso em: 5 nov. 2011.

CAVALCANTI, Lucas. **VRAPTOR 3**. [S.l.], 2009. Disponível em <http://www.infoq.com/br/articles/VRaptor3>. Acesso em: 05 nov. 2011.

DATE, C.J.. **What not How**: the business rules approach to application development. Boston: Addison-Wesley, 2000.

FERREIRA, Ricardo. **Drools Guvnor em 7 Passos.** [S.l.], 2010. Disponível em <http://architecture-journal.blogspot.com/2010/02/drools-guvnor-em-7-passos.html>. Acesso em: 12 out. 2011.

GRIFFIN, N.L.; LEWIS, F.D.. **A rule-based inference engine which is optimal and VLSI implementable**. Fairfax, VA, USA, 2002. Disponível em <http://www.cs.engr.uky.edu/~lewis/papers/inf-engine.pdf>. Acesso em: 5 nov. 2011.

JBOSS. **Drools 5 - The Business Logic integration Platform.** [S.l.], 2011a. Disponível em <http://www.jboss.org/drools>. Acesso em: 04 out. 2011.

JBOSS. **Drools Guvnor – Centralised Knowledge Repository.** [S.l.], 2011b. Disponível em <http://www.jboss.org/drools/drools-guvnor.html>. Acesso em: 12 out. 2011.

JBOSS. **Drools Expert User Guide.** [S.l.], 2011c. Disponível em <http://docs.jboss.org/drools/release/5.3.0.Final/drools-expert-docs/html_single/index.html>. Acesso em: 22 out. 2011.

KIJANOWSKI, Jarek. **JBoss Drools how-to:** Tuning Guvnor, part 1. [S.l.], 2008. Disponível em: < http://magazine.redhat.com/2008/08/12/jboss-drools-how-to-tuning-guvnor-part-1/>. Acesso em: 30 out. 2011.

LIU, Rafael. JBoss Drools 5. **Java Magazine**, [S.l.], n.88, p. 6-19, Fev. 2011.

LUYPAERT, Steffen. **Integrating Stuff - Setting up Drools Guvnor.** [S.l.], 2011. Disponível em <http://www.integratingstuff.com/2011/01/28/setting-up-drools-guvnor/>. Acesso em: 12 out. 2011.

OLIVIERI, Ricardo. **Implement business logic with the Drools rules engine**: Use a declarative programming approach to write your program's business logic. [S.l.], 2006. Disponível em: <http://public.dhe.ibm.com/software/dw/java/j-drools-pdf.pdf>. Acesso em: 06 out. 2011.

SAUDATE, Alexandre. **Business Rules - Rules hot deploy using Drools.** [S.l.], 2010. Disponível em < http://alesaudate.com/2010/05/30/hot-deploy-de-regras-utilizando-drools-

guvnor-parte-final/>. Acesso em: 5 nov. 2011.

SCHMIDT, Andrea Coimbra; NASCIMENTO, Elaine Letícia Camargo do; GORNI, Henrique Cesar. **NurDES – Nursing Diagnosis Expert System.** 2010. 194 f. Trabalho de Conclusão de Curso (Tecnologia em Sistemas de Informação) – Universidade Federal do Paraná.

SILVA, Kelli Aparecida Bez Batti da. **Análise Comparativa entre Programação Orientada a Objetos e Orientada a Aspectos**. 2005. 93 f. Trabalho de Conclusão de Curso (Bacharel em Sistemas de Informação) – Universidade Regional de Blumenau, Blumenau.

STRANDBERG, Niels Peter. **Rule-based Expert Systems – A practical example**: Artificial Intelligence and Intelligent Systems. [S.l.], 2005.

TIZZEI, Leonardo Pondian. **Uma infra-estrutura de suporte à evolução para repositórios de componentes.** 2007. 98 f. Programa de Mestrado em Ciência da Computação – Universidade Estadual de Campinas, Campinas.

APÊNDICE A – DETALHAMENTOS DOS CASOS DE USO

O Quadro 9 apresenta o caso de uso “Manter Advogado”.

Caso de uso – Manter Advogado
Ator: Administrador
Objetivo: Cadastrar Advogado
Pré-condições: Administrador precisa estar cadastrado no banco de dados.
Pós-condições: Uma conta de usuário foi inserida no sistema.

Cenário Principal:
1. Administrador seleciona a opção de cadastro de advogado
2. Sistema exibe formulário de cadastro de novos advogados
3. Administrador preenche as informações
4. Sistema valida os campos preenchidos
5. Sistema grava o cadastro do novo Advogado
6. Sistema exibe mensagem “Seu cadastro foi efetuado com sucesso!”

Cenário Alternativo:
No passo 4, caso houver campos obrigatórios não preenchidos:
4.1 Sistema apresenta mensagem “Favor preencher todos os campos obrigatórios”

Cenário Alternativo:
No passo 4, caso o login informado já estiver sendo utilizado por outro usuário:
4.1 Sistema apresenta mensagem “Este login está sendo utilizado por outro usuário.

Cenário Alternativo:
No passo 4, caso a senha informada for diferente do campo de confimação de senha:
4.1 Sistema apresenta mensagem “As senhas informadas não conferem”

Quadro 9 - Descrição do caso de uso Manter Advogado

O Quadro 10 apresenta o caso de uso "Logar no Sistema".

Caso de uso – Logar-se no Sistema
Ator: Administrador, Advogado
Objetivo: Logar-se no Sistema
Pré-condições: Usuário deve estar cadastrado no banco de dados.
Pós-condições: Usuário obtêm acesso ao sistema.

Cenário Principal:
1. Usuário preenche seu login e sua senha
2. Sistema valida os dados de login e senha do usuário
3. Sistema direciona o usuário para a página principal

Cenário Alternativo:
No passo 2, caso a senha ou usuário forem inválidos:
2.1 Sistema apresenta mensagem “usuário ou senha inválida”

Quadro 10 - Descrição do caso de uso Logar-se no Sistema

O Quadro 11 apresenta o caso de uso "`Manter pessoas`".

Caso de uso – Manter Pessoas
Ator: Advogado
Objetivo: Cadastrar Pessoas
Pré-condições: Advogado deve fazer login no sistema
Pós-condições: Advogado visualizou, editou, apagou ou cadastrou uma pessoa

Cenário Principal:
1. Advogado acessa o sistema
2. Advogado opta por editar, apagar ou cadastrar uma pessoa

Cenário Visualização:
2.1 Sistema mostra os registros das pessoas cadastradas para o Advogado

Cenário Edição:
2.1 Sistema mostra registros cadastrados
2.2 Advogado seleciona um registro para edição
2.3 Sistema mostra o nome, telefone, CPF, CNPJ e e-mail da pessoa para edição
2.4 Advogado altera registro e seleciona opção para atualizar os dados (nome, telefone, CPF, CNPJ e e-mail)
2.5 Sistema mostra os registros cadastrados com o registro alterado

Cenário Inclusão:
2.1 Sistema mostra registros cadastrados
2.2 Advogado inclui um novo registro
2.3 Sistema mostra os registros cadastrados

Cenário Exclusão:
2.1 Sistema mostra registros cadastrados
2.2 Advogado seleciona um registro para exclusão
2.3 Sistema exclui o registro e mostra os registros restantes

Quadro 11 - Descrição do caso de uso Manter pessoas

O Quadro 12 apresenta o caso de uso "`Manter tipos de participação`".

Caso de uso – Manter tipos de participação
Ator: Advogado
Objetivo: Cadastrar Tipos de Participação
Pré-condições: Advogado deve fazer login no sistema
Pós-condições: Advogado visualizou, editou, apagou ou cadastrou um tipo de participação

Cenário Principal:
1. Advogado acessa o sistema

2. Advogado opta por editar, apagar]ou cadastrar um tipo de participação

Cenário Visualização:
2.1 Sistema mostra os registros dos tipos de participação cadastrados para o Advogado

Cenário Edição:
2.1 Sistema mostra registros cadastrados
2.2 Advogado seleciona um registro para edição
2.3 Sistema mostra o nome do tipo de participação para edição
2.4 Advogado altera registro e seleciona opção para atualizar o nome do tipo de participação
2.5 Sistema mostra os registros cadastrados com o registro alterado

Cenário Inclusão:
2.1 Sistema mostra registros cadastrados
2.2 Advogado inclui um novo registro
2.3 Sistema mostra os registros cadastrados

Cenário Exclusão:
2.1 Sistema mostra registros cadastrados
2.2 Advogado seleciona um registro para exclusão
2.3 Sistema exclui o registro e mostra os registros restantes

Quadro 12 - Descrição do caso de uso Manter tipo de participação

O Quadro 13 apresenta o caso de uso "`Manter processo jurídico`".

Caso de uso – Manter processo jurídico
Ator: Advogado
Objetivo: Cadastrar Processo Jurídico
Pré-condições: Advogado deve fazer login no sistema
Pós-condições: Advogado visualizou, editou, apagou ou cadastrou um processo jurídico

Cenário Principal:
1. Advogado acessa o sistema
2. Advogado opta por editar, apagar ou cadastrar um processo jurídico

Cenário Visualização:
2.1 Sistema mostra os registros dos processos jurídicos cadastrados para o Advogado

Cenário Edição:
2.1 Sistema mostra registros cadastrados
2.2 Advogado seleciona um registro para edição
2.3 Sistema mostra número, assunto, data de abertura, data de fechamento e observação do processo jurídico para edição
2.4 Advogado altera registro e seleciona opção para atualizar os dados (número, assunto, data de abertura, data de fechamento e observação)
2.5 Sistema mostra os registros cadastrados com o registro alterado

Cenário Inclusão:

2.1 Sistema mostra registros cadastrados
2.2 Advogado inclui um novo registro
2.3 Sistema mostra os registros cadastrados

Cenário Exclusão:
2.1 Sistema mostra registros cadastrados
2.2 Advogado seleciona um registro para exclusão
2.3 Sistema exclui o registro e mostra os registros restantes

Quadro 13 - Descrição do caso de uso Manter processo jurídico

O Quadro 14 apresenta o caso de uso "`Manter participações`".

Caso de uso – Manter participações
Ator: Advogado
Objetivo: Cadastrar Participações
Pré-condições: Advogado deve fazer login no sistema
Pós-condições: Advogado visualizou, editou, apagou ou cadastrou uma participação

Cenário Principal:
1. Advogado acessa o sistema
2. Advogado opta por editar, apagar ou cadastrar uma participação

Cenário Visualização:
2.1 Sistema mostra os registros das participações cadastradas para o Advogado

Cenário Edição:
2.1 Sistema mostra registros cadastrados
2.2 Advogado seleciona um registro para edição
2.3 Sistema mostra código da pessoa, código do processo e data da participação para edição
2.4 Advogado altera registro e seleciona opção para atualizar os dados (código da pessoa, código do processo e data)
2.5 Sistema mostra os registros cadastrados com o registro alterado

Cenário Inclusão:
2.1 Sistema mostra registros cadastrados
2.2 Advogado inclui um novo registro
2.3 Sistema mostra os registros cadastrados

Cenário Exclusão:
2.1 Sistema mostra registros cadastrados
2.2 Advogado seleciona um registro para exclusão
2.3 Sistema exclui o registro e mostra os registros restantes

Quadro 14 - Descrição do caso de uso Manter participações

O Quadro 15 apresenta o caso de uso "`Manter andamentos`".

Caso de uso – Manter andamentos
Ator: Advogado
Objetivo: Cadastrar Andamentos
Pré-condições: Advogado deve fazer login no sistema
Pós-condições: Advogado visualizou, editou, apagou ou cadastrou um andamento

Cenário Principal:
1. Advogado acessa o sistema
2. Advogado opta por editar, apagar ou cadastrar um andamento

Cenário Visualização:
2.1 Sistema mostra os registros dos andamentos cadastrados para o Advogado

Cenário Edição:
2.1 Sistema mostra registros cadastrados
2.2 Advogado seleciona um registro para edição
2.3 Sistema mostra código do processo, descrição, data inicial e data final do andamento para edição
2.4 Advogado altera registro e seleciona opção para atualizar os dados código do processo, descrição, data inicial e data final)
2.5 Sistema mostra os registros cadastrados com o registro alterado

Cenário Inclusão:
2.1 Sistema mostra registros cadastrados
2.2 Advogado inclui um novo registro
2.3 Sistema mostra os registros cadastrados

Cenário Exclusão:
2.1 Sistema mostra registros cadastrados
2.2 Advogado seleciona um registro para exclusão
2.3 Sistema exclui o registro e mostra os registros restantes

Quadro 15 - Descrição do caso de uso Manter andamentos

O Quadro 16 apresenta o caso de uso "`Manter audiências`".

Caso de uso – Manter audiências
Ator: Advogado
Objetivo: Cadastrar Audiências
Pré-condições: Advogado deve fazer login no sistema
Pós-condições: Advogado visualizou, editou, apagou ou cadastrou uma audiência

Cenário Principal:
1. Advogado acessa o sistema
2. Advogado opta por editar, apagar ou cadastrar uma audiência

Cenário Visualização:
2.1 Sistema mostra os registros das audiências cadastradas para o Advogado

Cenário Edição:

2.1 Sistema mostra registros cadastrados
2.2 Advogado seleciona um registro para edição
2.3 Sistema mostra a código do processo, data, local e descrição da audiência para edição
2.4 Advogado altera registro e seleciona opção para atualizar os dados (código do processo, data, local e descrição)
2.5 Sistema mostra os registros cadastrados com o registro alterado

Cenário Inclusão:
2.1 Sistema mostra registros cadastrados
2.2 Advogado inclui um novo registro
2.3 Sistema mostra os registros cadastrados

Cenário Exclusão:
2.1 Sistema mostra registros cadastrados
2.2 Advogado seleciona um registro para exclusão
2.3 Sistema exclui o registro e mostra os registros restantes

Quadro 16 - Descrição do caso de uso Manter audiências

APÊNDICE B – DICIONÁRIO DE DADOS

Este apêndice apresenta a descrição detalhada das entidades da modelagem de banco de dados previstas (Figura 15). Os tipos de dados de cada campo são descritos a seguir:

a) varchar - tipo de campo para armazenamento de strings de caracteres e seu tamanho é definido em bytes com largura variável, os valores entre parênteses definem o comprimento máximo em bytes de caracteres;
b) int - tipo de campo para armazenamento de números inteiros;
c) date - tipo de campo para armazenamento de datas;
d) text - tipo de campo para armazenamento de grandes strings ou binários.

No Quadro 17 pode-se observar o dicionário de dados da tabela “usuario”.

Tabela: Usuario				
Campo	**Tipo**	**Descrição**	**Opcional**	**Chave**
id_usuario	int(11)	Id do usuário	Não	Primária
nm_usuario	varchar(45)	Nome do usuário	Sim	
dt_nascimento	date	Data de nascimento	Sim	
ds_login	varchar(45)	Login	Não	
ds_senha	varchar(45)	Senha	Sim	
fl_administrador	varchar(1)	Indicador de administrador	Não	

Quadro 17 - Dicionário de dados da tabela "usuario"

No Quadro 18 pode-se observar o dicionário de dados da tabela “pessoa”.

Tabela: Pessoa				
Campo	**Tipo**	**Descrição**	**Opcional**	**Chave**
id_pessoa	int(11)	Id da Pessoa	Não	Primária
nm_pessoa	varchar(45)	Nome da Pessoa	Não	
nr_telefone	varchar(10)	Telefone da Pessoa	Sim	
fl_pessoa_fisica	varchar(1)	Indicador de Pessoa Física	Não	
nr_cpf	varchar(14)	Número CPF	Sim	
nr_cnpj	varchar(18)	Número CNPJ	Sim	
ds_email	varchar(45)	E-mail da Pessoa	Sim	
id_usuario	int(11)	Id do usuário	Não	Estrangeira

Quadro 18 - Dicionário de dados da tabela "pessoa"

No Quadro 19 pode-se observar o dicionário de dados da tabela “tipo_participacao”.

Tabela: Tipo_participacao				
Campo	**Tipo**	**Descrição**	**Opcional**	**Chave**
id_tipo_participacao	int(11)	Id do Tipo de Participação	Não	Primária
nm_tipo	varchar(45)	Nome do Tipo de Participação	Não	
id_usuario	int(11)	Id do usuário	Não	Estrangeira

Quadro 19 - Dicionário de dados da tabela "tipo_participacao"

No Quadro 20 pode-se observar o dicionário de dados da tabela “processo”.

Tabela: Processo				
Campo	**Tipo**	**Descrição**	**Opcional**	**Chave**
id_processo	int(11)	Id do Processo	Não	Primária
nr_processo	varchar(45)	Número do Processo	Não	
ds_assunto	text	Assunto do Processo	Não	
dt_abertura	date	Data de Abertura do Processo	Sim	
dt_fechamento	date	Data de Fechamento Processo	Sim	
ds_observacao	text	Observação do Processo	Sim	
id_usuario	int(11)	Id do usuário	Não	Estrangeira

Quadro 20 - Dicionário de dados da tabela "processo"

No Quadro 21 pode-se observar o dicionário de dados da tabela “participacao”.

Tabela: Participacao				
Campo	**Tipo**	**Descrição**	**Opcional**	**Chave**
id_participacao	int(11)	Id da Participação	Não	Primária
id_pessoa	int(11)	Id da Pessoa	Não	Primária
id_processo	int(11)	Id do Processo	Não	Primária
id_tipo_participacao	int(11)	Id do Tipo de Participação	Não	Estrangeira
dt_associacao	date	Data da Participação	Não	
id_usuario	int(11)	Id do Usuário	Não	Estrangeira

Quadro 21 - Dicionário de dados da tabela "participacao"

No Quadro 22 pode-se observar o dicionário de dados da tabela “audiencia”.

Tabela: Audiencia				
Campo	**Tipo**	**Descrição**	**Opcional**	**Chave**
id_audiencia	int(11)	Id da Audiência	Não	Primária
id_processo	int(11)	Id do Processo	Não	Estrangeira
dt_audiencia	date	Data da Audiência	Não	
ds_local	varchar(75)	Local da Audiência	Sim	
ds_audiencia	text	Descrição da Audiência	Não	
id_usuario	int(11)	Id do Usuário	Não	Estrangeira

Quadro 22 - Dicionário de dados da tabela "audiencia"

No Quadro 23 pode-se observar o dicionário de dados da tabela “andamento”.

Tabela: Andamento				
Campo	**Tipo**	**Descrição**	**Opcional**	**Chave**
id_andamento	int(11)	Id do Andamento	Não	Primária
id_processo	int(11)	Id do Processo	Não	Estrangeira
ds_andamento	varchar(45)	Descrição do Andamento	Não	

dt_inicial	date	Data Inicial do Andamento	Não	
dt_final	date	Data Final do Andamento	Não	
id_usuario	int(11)	Id do Usuário	Não	Estrangeira

Quadro 23 - Dicionário de dados da tabela "andamento"