



UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO

ESCOLA DE INFORMÁTICA APLICADA

CURSO DE BACHARELADO EM SISTEMAS DE
INFORMAÇÃO

Desenvolvimento de uma API para busca e edição de contextos no Logtell

Victor Gonçalves Boaventura

Orientador:

Angelo Ernani Maia Ciarlini

Desenvolvimento de uma API para busca e edição de contextos no Logtell

Victor Gonçalves Boaventura

Projeto de Graduação
apresentado à Escola de Informática
Aplicada da Universidade Federal do
Estafo do Rio de Janeiro (UNIRIO) para
obtenção do título de Bacharel em
Sistemas de Informação.

Rio de Janeiro - RJ

Agosto/2013

Desenvolvimento de uma API para busca e edição de contextos no Logtell

Aprovado em: _____/_____/_____

Banca Examinadora:

Angelo Ernani Maia Ciarlini

Sean Wolfgang Matsui Siqueira

Leila Cristina Vasconcelos de Andrade

O autor deste projeto autoriza a ESCOLA DE INFORMÁTICA APLICADA da UNIRIO a divulgá-lo, no todo ou em parte, resguardando os direitos autorais conforme legislação vigente.

Rio de Janeiro, _____ de _____ de _____.

Victor Gonçalves Boaventura

Agradecimentos

A meus amigos e familiares pelo apoio.

Ao meu orientador, Angelo Ciarlini,
pela paciência e dedicação.

À professora Leila Andrade, pelo
estímulo necessário no momento certo.

A todo corpo docente da UNIRIO pela
excelência no ensino e pelo aprendizado
proporcionado.

SUMÁRIO

| | |
|--|-----------|
| 1. INTRODUÇÃO..... | 1 |
| 1. INTRODUÇÃO | |
| 1.a. Motivação..... | 1 |
| 1.b. Proposta..... | 2 |
| 1.c. Estrutura..... | 2 |
| 2. FUNDAMENTAÇÃO | 4 |
| 2. FUNDAMENTAÇÃO | |
| 2.a. Storytelling Interativo | 4 |
| 2.b. Logtell..... | 5 |
| 2.c. Dramatização não determinística de eventos..... | 8 |
| 3. PROPOSTA | 10 |
| 3. PROPOSTA | |
| 3.a. Descrição do problema | 10 |
| 3.b. Premissas | 13 |
| 3.c. Tecnologias..... | 15 |
| 3.d. Arquitetura..... | 16 |
| 4. IMPLEMENTAÇÃO | 19 |
| 4. IMPLEMENTAÇÃO | |
| 4.a. Camada de armazenamento..... | 19 |
| 4.b. Camada de processamento | 30 |
| 4.c. Camada de apresentação..... | 35 |
| 5. RESULTADOS | 51 |
| 5. RESULTADOS | |
| 5.a. Expressões booleanas de um estado..... | 53 |
| 5.a. Parâmetros de micro ação | 51 |
| 6. CONCLUSÃO | 61 |
| 7. REFERÊNCIA BIBLIOGRÁFICA | 62 |
| APÊNDICE 1. MAPEAMENTO DAS ENTIDADES | 63 |

RESUMO

Neste trabalho é definido o desenvolvimento de uma API (*Application Programming Interface*) para visualização e edição de contextos no sistema de *storytelling* para TV interativa Logtell 2. O propósito desta API é permitir futuramente a construção de uma ferramenta autoral, que proporcione escalabilidade na produção de histórias para o Logtell 2.

Palavras-chave: Storytelling Interativo; Ferramenta Autoral; Desenvolvimento; API.

1. INTRODUÇÃO

O objetivo deste capítulo é contextualizar as motivações e ideias abordadas por este trabalho, bem como apresentar a estrutura dos capítulos.

a. Motivação

Com o avanço da tecnologia e o surgimento da TV Digital, o conceito de TV Interativa vem se tornando cada vez mais uma realidade. Dentre as aplicações para TV Interativa, destacam-se as aplicações de *storytelling* interativo, que permitem a um usuário interferir em uma história que lhe é contada. Estas aplicações devem atender um conjunto de requisitos, como coerência e diversidade de histórias, fluxo contínuo na narrativa, e escalabilidade na produção e transmissão de conteúdo.

O Logtell – detalhado no capítulo 2 – é um sistema de storytelling para TV Interativa que já possui a capacidade de gerar histórias de forma coerente e interessante, além de dramatizar estas histórias por meio de animação gráfica 3D. Além disso, o Logtell possui mecanismos que conferem à dramatização diversidade, controle do tempo, e possibilidades de interação do usuário.

Porém a criação dos eventos que compõe uma história do Logtell exige um grande esforço autoral para atender aos requisitos de diversidade da dramatização. Falta à estrutura atual do Logtell uma interface que permita a criação destes eventos com agilidade e sem a necessidade de conhecimento técnico. Para que o sistema obtenha sucesso, é de suma importância a capacidade de geração de conteúdo em larga escala, tendo em vista que o sistema foi concebido visando atender múltiplos usuários

simultâneos, seja interagindo com histórias diversas, ou compartilhando suas interações na mesma história.

Desta forma, a motivação deste trabalho é a definição de uma API que permita a criação de uma interface para que autores de histórias interativas possam criar conteúdo para o Logtell de forma prática e amigável.

b. Proposta

Partindo do princípio de que o processo de criação de uma interface gráfica para autoria de histórias interativas exigiria um profundo estudo de design gráfico e usabilidade para que esta interface fosse construída adequadamente, além do conhecimento técnico necessário para incorporar a mesma ao sistema existente, tornou-se evidente a necessidade de incorporar ao Logtell uma forma padronizada de inserir e editar o contexto de histórias interativas. Sendo assim, interfaces de diferentes naturezas poderiam fazer uso deste protocolo de comunicação com o Logtell, reduzindo drasticamente o desafio técnico para criação destas.

Uma API (Application Programming Interface) é um protocolo utilizado para que diferentes componentes de software se comuniquem. Desta forma, quaisquer interfaces construídas como ferramenta autoral para o Logtell podem ser concebidas como componentes isolados de software, que se comunicariam com o sistema existente através desta API.

Além disso, afim de demonstrar a capacidade desta API, será proposta uma interface simples para visualização destas histórias, baseada em formulários web.

c. Estrutura

O trabalho está dividido da seguinte forma:

- **Capítulo 1:** Capítulo contendo a introdução do trabalho. Este capítulo apresenta a motivação e as ideias por trás deste trabalho.
- **Capítulo 2:** Capítulo contendo a fundamentação teórica do trabalho. Este capítulo apresenta conceitos relacionados a storytelling interativo, Logtell e dramatização não determinística de eventos.
- **Capítulo 3:** Capítulo contendo a proposta do trabalho. Este capítulo detalha a descrição do problema, as premissas para elaboração da API, as tecnologias utilizadas, e a arquitetura proposta.
- **Capítulo 4:** Capítulo contendo a implementação do trabalho. Este capítulo detalha os pormenores da implementação das diferentes camadas da API.
- **Capítulo 5:** Capítulo contendo o resultado do trabalho. Este capítulo exemplifica os benefícios obtidos a partir da elaboração da API.
- **Capítulo 6:** Capítulo contendo a conclusão do trabalho.
- **Capítulo 7:** Capítulo contendo as referências bibliográficas utilizadas por este trabalho.

2. FUNDAMENTAÇÃO

Este capítulo tem a função de apresentar o contexto do projeto. Como se trata da definição de uma API que possibilite a construção de uma interface gráfica e amigável para a definição de eventos no sistema de storytelling interativo Logtell, será detalhado o conceito de storytelling interativo (seção 2.a.) e o sistema em questão (seção 2.b.).

Será também de grande importância para a compreensão deste projeto o entendimento da técnica de dramatização não-determinística de eventos utilizada pelo Logtell (seção 2.c.).

A implementação da dramatização não-determinística de eventos tornou o Logtell mais flexível e adequado para um ambiente de TV interativa. Contudo, foi justamente essa funcionalidade que tornou evidente a necessidade de esforço autoral na especificação de eventos e criação das histórias interativas. A inexistência de uma interface para este fim limita as possibilidades de uso do Logtell, já que autores precisariam inserir manualmente no banco de dados a informação que especifica esses eventos.

a. Storytelling Interativo

O Storytelling interativo é, essencialmente, uma forma de entretenimento digital onde usuários têm a capacidade de influenciar ativamente o decorrer de uma história enquanto ela está sendo contada.

Ao contrário de jogos digitais, onde a história serve como catalisador para a motivação do jogador e como pano de fundo para contextualizar os desafios enfrentados, no storytelling interativo o principal objetivo é gerar narrativas que sejam ao mesmo

tempo coerentes e interessantes (DÓRIA, 2009), permitindo que o usuário possa intervir na história sem violar as regras do gênero (CIARLINI; POZER *et al*, 2005).

b. Logtell

O Logtell (CIARLINI; POZER *et al*, 2005, POZZER, 2005) é um sistema de storytelling interativo para iTV, originalmente concebido com uma perspectiva de terceira pessoa, e parte do princípio de que as convenções que definem um gênero podem ser logicamente modeladas, enquanto permite ao usuário a capacidade de dirigir a narrativa de diversas formas, como escolher alternativas para o futuro, retroceder a história a pontos anteriores e forçar o acontecimento de eventos. As intervenções do usuário, contudo, estão presas às definições do modelo lógico do gênero, de forma que intervenções que não respeitem esse modelo são descartadas.

A principal mudança do Logtell 2 (CAMANHO *et al*, 2009, CAMANHO, 2009), desenvolvido a partir de sua primeira versão, é o paralelismo na geração e dramatização dos enredos. Para que isto fosse possível, foi implementada uma arquitetura no modelo cliente-servidor, de acordo com a Figura 1 a seguir (DÓRIA, 2009):

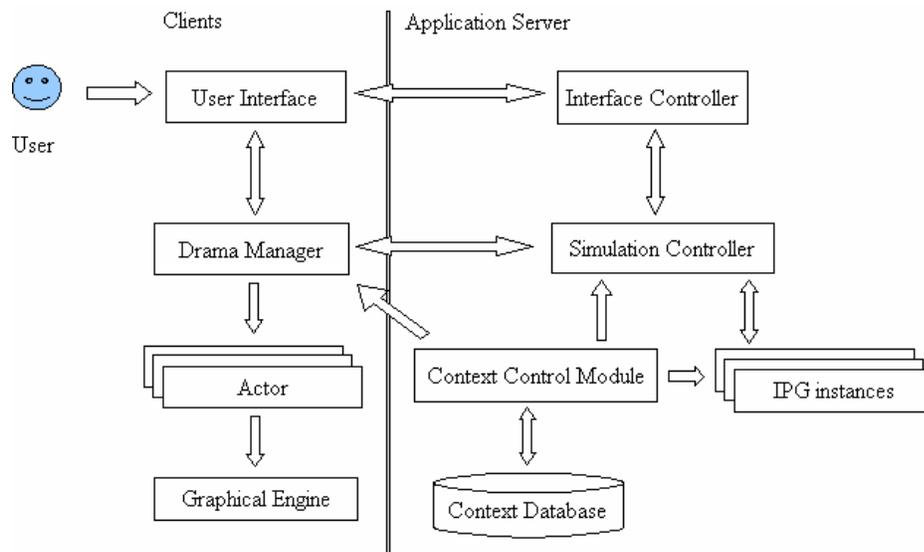


Figura 1: Arquitetura do Logtell 2

O Contexto das histórias é armazenado em um banco de dados, e é acessado pelo Context Control Module no servidor de aplicação durante a geração e dramatização de enredos. O contexto contém a descrição do gênero – cujas regras regem a geração de histórias – e o estado inicial dos personagens e do ambiente. O gênero é descrito da seguinte forma:

- Um conjunto de operações básicas parametrizadas, com pré-condições e pós-condições, que especificam logicamente o conjunto de eventos que podem ocorrer;
- Um conjunto de regras de inferência, especificado em um formalismo de lógica temporal, que definem as situações que levam ao surgimento de objetivos dos personagens ou da história como um todo;
- Uma biblioteca de planos, correspondentes a sequências de operações que levem ao alcance de objetivos específicos, organizada em hierarquias de composição e especialização.

Os enredos são gerados por instâncias do Interactive Plot Manager (IPG) (CIARLINI, 1999) no servidor de aplicação como uma sequência de capítulos. Cada um

destes capítulos corresponde a um ciclo onde objetivos são inferidos e alcançados a partir de planejamento.

Da sequência de capítulos é gerado um enredo. Caso o usuário não faça nenhuma intervenção, os objetivos são inferidos e os capítulos são apresentados de forma contínua e o usuário assiste à história como na TV convencional. Contudo, conforme mencionado anteriormente, a história pode sofrer intervenções do usuário. Essas interações são divididas em fortes e fracas:

- Nas interações fortes, o usuário ativamente indica que deseja a ocorrência de determinado evento ou situação no próximo capítulo, selecionados a partir de sugestões apresentadas automaticamente pelo sistema;
- Nas interações fracas, o usuário força a história a retornar a um capítulo anterior para que uma alternativa para a história seja automaticamente gerada e dramatizada, ou para assistir novamente ao mesmo capítulo com a possibilidade de alterar a narrativa a partir daquele ponto através de interações fortes.

Quando o usuário intervém na história, o planejador tenta incorporar essas interações ao enredo, sempre verificando a coerência lógica dessas intervenções solicitadas pelo usuário. Caso a interação não seja consistente com as regras que definem o gênero, a mesma é desconsiderada.

O servidor de aplicação, através do Simulation Controller, se comunica com o Drama Manager dos clientes para informar os próximos eventos a serem dramatizados; para receber solicitações de interação do usuário e incorporar os mesmos, caso seja possível; para selecionar sugestões de intervenções viáveis; para obter os próximos eventos a serem dramatizados do IPG; e para sincronizar as dramatizações com a geração do enredo.

O usuário interage com o sistema através da interface com o usuário (User Interface) que passa as interações realizadas para o Interface Controller no servidor. O Drama Manager do cliente solicita ao já descrito Simulation Controller o próximo

evento a ser dramatizado e controla as instâncias dos atores que são dramatizados em ambiente 3D em execução no Graphical Engine, o motor de jogos 3D.

Em execução no servidor, o Interface Controller centraliza as sugestões feitas pelos múltiplos clientes. Em situações onde vários usuários compartilham a mesma história, as interações são selecionadas de acordo com o número de clientes que solicitam as mesmas.

O Logtell 2 é capaz de gerar e dramatizar histórias simultaneamente, mantendo a coerência e diversidade dos enredos enquanto permite aos usuários a capacidade de interação com as histórias em diversos níveis. Além disso, a arquitetura do sistema leva em consideração a possibilidade de múltiplos usuários simultaneamente compartilharem o uso do sistema interagindo com histórias coletivas ou individuais.

c. Dramatização não determinística de eventos

Para possibilitar maior variedade na dramatização dos eventos que formam cada capítulo, permitindo que o Logtell seja mais adequado a um ambiente de TV interativa, o sistema possui um modelo de controle de dramatização (DÓRIA, 2009), que introduz um nível de não-determinismo na dramatização de cada evento, de forma que cada ator possa executar ações cujos efeitos são variados. Além disso, o não-determinismo foi estendido para que o usuário possa, se desejar, interferir no resultado dessas ações, inclusive contando com a possibilidade de retardar ou acelerar a conclusão de um evento.

Para que cada evento seja concluído da maneira esperada pelo enredo, mecanismos de controle foram estabelecidos garantindo que os efeitos esperados sejam gerados. Para este fim, cada evento é modelado como um autômato não-determinístico, e são utilizadas técnicas de planejamento baseado em verificação de modelos (DÓRIA, 2009) para gerar políticas que determinam as ações a serem executadas quando o evento se encontra em determinado estado. Como vários estados podem ser alcançados a partir

de um estado atual, essas políticas têm a função de conduzir a dramatização a um estado final desejado.

No que tange ao controle da dramatização, alguns requisitos se mostraram necessários:

- Controle do tempo: Para que um fluxo contínuo seja mantido, é fundamental que seja possível ter controle da duração de cada evento. Se o mesmo ocorrer rápido demais, pode-se perder o sincronismo entre a geração do enredo – uma tarefa complexa – e a dramatização do mesmo. Por outro lado, uma dramatização demasiadamente longa pode entediar o usuário. Idealmente, o tempo da dramatização de um evento precisa ser o suficiente para que ocorra de forma natural e transmita a carga dramática planejada pelo autor.
- Diversidade com coerência: A diversidade das histórias geradas e apresentadas é um dos pontos fundamentais para o funcionamento adequado do sistema em um ambiente de TV interativa. Desta forma, a diversidade na dramatização é importante para que este objetivo seja atingido. Quando uma mesma situação pode ser contada de diversas formas, mesmo que o final de cada evento seja mantido, o número de narrativas distintas que podem ser experimentadas pelos usuários torna-se maior.
- Possibilidades de interação: Apesar de o sistema já levar originalmente em consideração a possibilidade de o usuário intervir nas situações e eventos que devem ocorrer no capítulo seguinte, a possibilidade de interação direta com o evento que está sendo dramatizado permite um maior nível de imersão do usuário. Para que isso seja possível, a dramatização deve oferecer flexibilidade sem que a consistência do enredo seja perdida.
- Eficiência na dramatização: Como o fluxo da narrativa precisa ser contínuo, a implementação de um modelo flexível para a dramatização não pode resultar na diminuição da eficiência. Para isso, torna-se necessária a existência de mecanismos para a geração prévia de alternativas de controle, evitando a necessidade de planejamento da dramatização em tempo real.

3. PROPOSTA

Neste capítulo será detalhado o protocolo para edição e validação de contextos para o Logtell. Este protocolo será utilizado por uma interface gráfica simples para visualização do contexto das histórias, porém será flexível o suficiente para interfaces mais complexas que tratem do problema de inclusão e edição de histórias no sistema.

A interface gráfica de visualização de contextos será incluída na API apenas como forma de exemplificar seu uso.

a. Descrição do problema

O sistema Logtell, atualmente, já possibilita a geração e a dramatização e enredos de forma variada e coerente a partir de enredos logicamente modelados, além de permitir ao espectador um bom grau de interatividade com essa história enquanto a mesma é dramatizada.

O modelo lógico de um enredo, referido a partir daqui como “contexto”, define as possíveis sequências de eventos que compõe esse contexto. Cada evento é modelado como um autômato finito não-determinístico, de forma que a dramatização desse evento pelo motor gráfico 3D ocorra de forma variada, e possibilite uma maior interação do espectador com a história.

O autômato que determina a modelagem lógica de um evento é especificado a partir de um conjunto de parâmetros, um conjunto de estados, e um conjunto de ações. Este autômato determinará as ações que serão executadas sucessivamente até que um estado final seja alcançado. No modelo atual, assume-se que cada evento tem somente um estado final e um estado inicial, para que a coerência do enredo seja mantida. Para que a dramatização de eventos possa ter múltiplos estados finais, a geração de enredos,

que determina os eventos a serem dramatizados, precisa ser capaz de lidar com esse não determinismo.

A definição dos eventos é persistida em um complexo banco de dados, tornando o esforço autoral dependente de conhecimentos de programação e do esquema relacional correspondente à especificação de contextos. O modelo de dados Entidade-Relacionamento que define um evento está representado no diagrama da Figura 2 (DÓRIA, 2009):

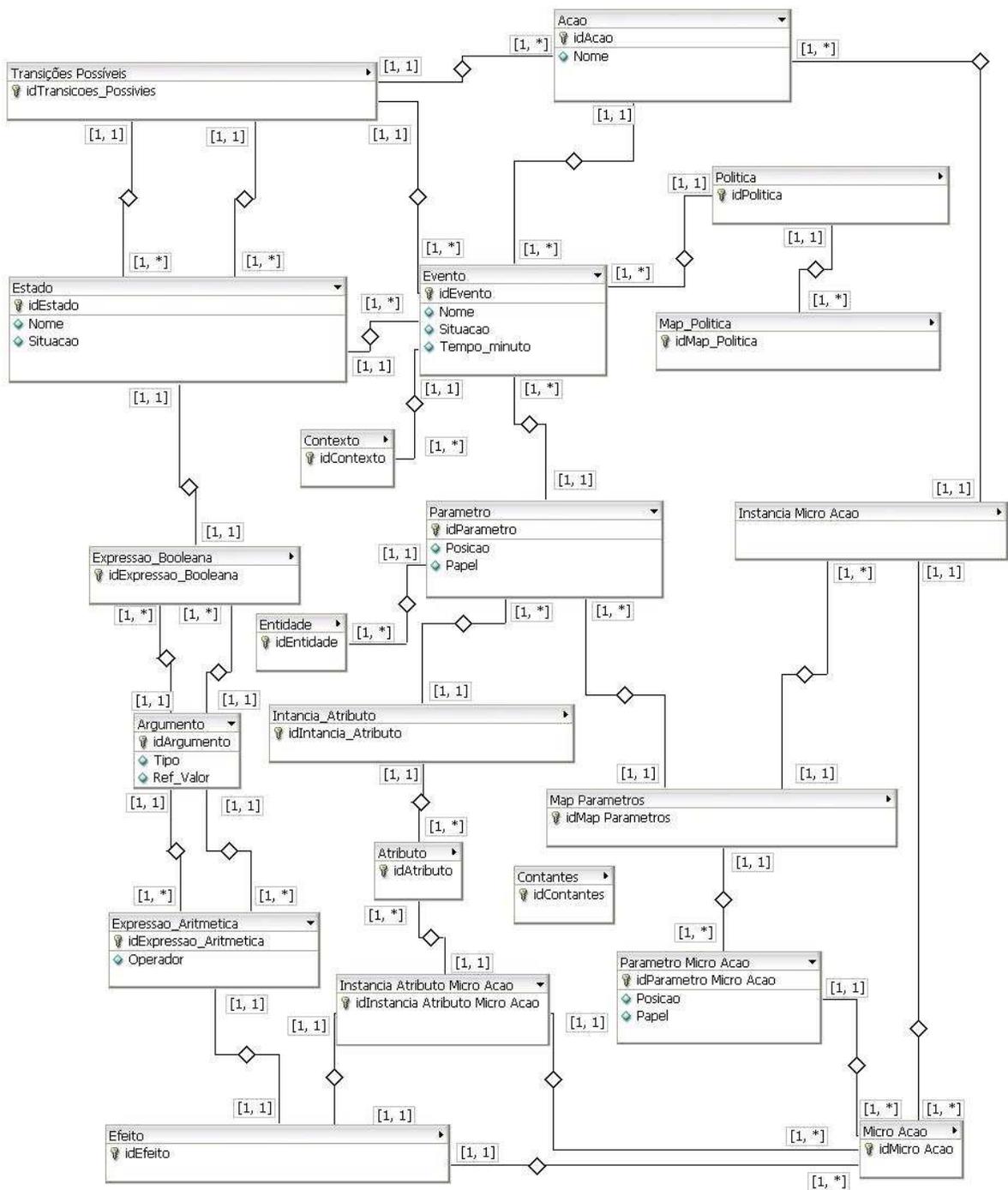


Figura 2: Modelo de dados Entidade-Relacionamento relativo à especificação de contextos.

Os parâmetros são atributos que possuem valores e representam entidades como personagens ou lugares. Estes atributos funcionam como variáveis de controle, sobre os

quais são aplicadas fórmulas lógicas para que seja definido o estado atual da dramatização. Ações realizam operações matemáticas sobre os parâmetros, alterando seu valor e efetivamente modificando o estado atual do autômato.

Os estados são os vértices do grafo que especifica o autômato. A especificação do contexto deve procurar garantir que todas as possibilidades de estados possam ser atingidas. Além disso, para cada estado é indicado também o quão próximo o mesmo se encontra do estado final, para que se tenha um controle mais firme sobre o tempo de dramatização.

As ações são as arestas do grafo que especifica o autômato, e representam as transações não determinísticas entre os estados. Sendo assim, uma ação pode levar o autômato de um estado a vários possíveis estados, através da modificação de parâmetros do evento. Elas são implementadas por meio de fluxos de micro-ações ordenadas aos atores virtuais. Esses fluxos podem conter ciclos, mas devem ser sempre finitos, e uma micro-ação pode ser habilitada ou desabilitada dependendo de condições sobre variáveis do evento a ser dramatizado. Além disso, esses fluxos podem conter pontos de escolha onde uma micro-ação dentre várias pode ser executada.

A partir da definição desse autômato, políticas são automaticamente geradas, definindo quais ações podem – ou devem – ser executadas sob certas circunstâncias, a fim de atingir um determinado estado de acordo com as necessidades atuais da dramatização.

b. Premissas

Para a elaboração da API e de uma ferramenta autoral para o sistema Logtell, serão consideradas as seguintes premissas:

- **Ferramenta para autores:** Enquanto o Logtell já possui uma aplicação cliente focada na dramatização e interatividade dos espectadores com as histórias geradas, o mesmo não possui uma interface para autores editarem os

contextos dessas histórias. Uma interface bem sucedida precisa ser capaz de realizar buscas e alterações de contextos na base de dados do Logtell, bem como fazer as devidas validações sobre esses dados.

- **Interface web-based:** O Logtell é baseado numa arquitetura cliente-servidor. Nesse modelo, múltiplos clientes se conectam em um servidor de aplicação (ou em um pool de servidores). Os clientes são, em geral, responsáveis pela interação com o usuário e pela dramatização das histórias, enquanto o servidor pela geração e controle das histórias apresentadas para os clientes. Sendo assim, parece adequado que quaisquer ferramentas de edição façam uso de uma interface web, para que autores remotos possam editar simultaneamente diferentes contextos sem interferir no funcionamento da aplicação cliente.
- **Arquitetura independente:** A interface web para autores deve ser independente da estrutura física e da arquitetura do Logtell. Para que seja garantida a escalabilidade e a confiabilidade dos servidores responsáveis pela geração e controle das histórias, deve ser possível (mas não necessário) que a ferramenta autoral funcione em servidores distintos, e apenas compartilhe acesso ao mesmo banco de dados.
- **Controle de usuários:** Levando em consideração a possibilidade de múltiplos autores utilizarem o sistema para edição de contextos, torna-se necessário um controle de usuários, para que um contexto não seja editado por autores diferentes ao mesmo tempo.
- **Manutenção do esquema do BD:** Como o Logtell, hoje em dia, já opera sobre o banco de dados com um esquema definido, alterações nas tabelas desse banco devem ser evitadas, na medida do possível, para que não haja impacto no funcionamento do sistema como ele é hoje. Desta forma, a API deve ser adaptada ao esquema existente do banco de dados.
- **Constante evolução:** Como o Logtell mostrou ao longo do tempo passar por um considerável número de evoluções, é importante que a API para edição de contextos seja flexível o suficiente para que seja facilmente estendida ou

adaptada para novos cenários, sem que para isso precise ser inteiramente redesenhada.

c. Tecnologias

Levando em consideração as premissas já mencionadas para o desenvolvimento da API e de uma ferramenta autoral, definiremos as tecnologias a serem utilizadas no desenvolvimento:

- **Java:** Linguagem de programação orientada a objeto, concorrente, largamente utilizada em aplicações web cliente-servidor. Possui um grande número de frameworks e bibliotecas voltado para o desenvolvimento de aplicações web, além de ser a linguagem originalmente utilizada para o desenvolvimento do Logtell, o que facilitaria sua manutenção.
- **Jboss Application Server:** JBoss AS é um servidor de aplicação que implementa a plataforma JEE (Java Enterprise Edition), e pode ser utilizado em quaisquer sistemas operacionais que suportem Java. É o servidor de aplicação já utilizado pelo Logtell, e tornaria mais fácil a implantação da API.
- **JEE:** Plataforma de programação para servidores de aplicação que define especificações para uma série de funcionalidades. O fornecedor de software que implementa essa plataforma precisa seguir estas especificações para que seja compatível com o padrão JEE.
- **EAR:** Enterprise Archive é um formato de arquivo especificado pela plataforma JEE que permite a implantação simultânea de vários módulos da mesma aplicação de forma coerente.
- **EJB:** Enterprise Java Bean é uma das especificações definidas na plataforma JEE, buscando padronização ao implementar regras de negócio no back-end de aplicações distribuídas, tendo como principal benefício a integridade

transacional nesse tipo de ambiente – o que seria fundamental num cenário de alterações concorrentes.

- **JSF:** Java Server Faces é um framework web-based que visa simplificar o desenvolvimento de interface com o usuário, através bibliotecas de componentes de interface que podem ser utilizados na construção de páginas, formas de associar eventos no lado cliente com controles do lado servidor, suporte à tecnologia AJAX, entre outras características.
- **JPA:** Java Persistence API é uma interface que faz parte da plataforma JEE, e define padrões para a persistência de objetos em um Banco de Dados relacional, como o utilizado pelo Logtoll. Como JPA é apenas uma interface, a implementação escolhida é o Hibernate.
- **Jboss Seam:** Framework baseado na plataforma JEE para desenvolvimento de aplicações web, e integra tecnologias como AJAX, JSF, JPA e EJB, adicionando algumas funcionalidades a essas tecnologias.
- **AJAX:** Acrônimo para Asynchronous JavaScript and XML, trata-se de um conjunto de técnicas utilizadas no lado cliente de uma aplicação web para que seja feita comunicação assíncrona com o servidor, possibilitando requisições que são respondidas sem interferir diretamente no conteúdo e comportamento da página existente.
- **JUnit:** Framework de suporte a testes unitários, que será utilizado para validar diversas funcionalidades de validação e edição da API.

d. Arquitetura

A partir da definição dos requisitos e das premissas, e da escolha das tecnologias a serem utilizadas, será proposta uma arquitetura para o desenvolvimento da API.

Pelas características da API, a opção de uma arquitetura em três camadas parece ser a mais apropriada, por ser uma arquitetura cliente-servidor onde a apresentação, o

processamento das regras da aplicação, e as funcionalidades de gerenciamento de dados, estão logicamente separadas.

Levando em consideração a constante evolução do Logtell, esse tipo de arquitetura permite flexibilidade para a adição de novas camadas de acordo com a necessidade, ou mesmo a modificação de uma camada existente sem que para isso toda a API precise ser reconstruída.

No que se referem à API, as camadas estariam dispostas conforme o diagrama da Figura 3:

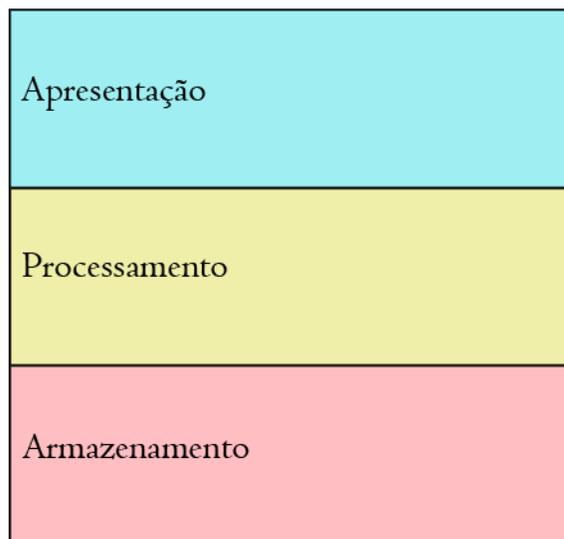


Figura 3: Definição das camadas que compõe a arquitetura da API.

A camada de apresentação consiste na interface web para visualização dos contextos, utilizando tecnologias como JSF e Ajax. Nesta camada estarão reunidas as páginas e arquivos de configuração necessários para o acesso web, bem como a parte da aplicação responsável por atender às requisições web e lidar com a exibição de dados.

A camada de processamento é a parte da API responsável pelo processamento das regras de validação de contextos e controle transacional, e é acessada diretamente pela camada de apresentação. Neste modelo, quaisquer novas interfaces que forem

adicionadas para edição de contextos, devem utilizar os métodos disponíveis nessa camada para buscar, inserir, ou alterar dados de contextos.

A camada de armazenamento é a parte da API responsável pela comunicação com o banco de dados, fazendo uso de JPA para o mapeamento objeto-relacional das entidades modeladas no banco e contando com métodos acessados pela camada de processamento para realizar as operações necessárias para o armazenamento de dados.

A escolha deste modelo de arquitetura está intimamente associada à escolha da plataforma JEE, pois a mesma assume uma abordagem multicamada na definição das APIs que compõe a plataforma. Estas camadas são logicamente separadas e possuem baixo acoplamento entre si, possibilitando a desejada flexibilidade na sua implementação.

Detalhes da implementação são discutidos no capítulo a seguir. Esse detalhamento explicita como as APIs da plataforma JEE se relacionam com as camadas da API para edição de contextos do Logtell.

4. IMPLEMENTAÇÃO

Os detalhes da implementação da API para edição e validação de contextos para o Logtell serão especificados neste capítulo. Para o melhor entendimento, cada uma das camadas que compõe a API será detalhada separadamente, tendo em vista que estas são independentes entre si.

Primeiramente será detalhada a camada de armazenamento (seção 4.a.), onde será explicitado como foi feito o mapeamento objeto-relacional para que a API possa acessar o banco de dados através de JPA e a partir de quais classes e métodos esse acesso é feito.

A seguir será detalhada a camada de processamento (seção 4.b.), responsável por acessar a camada de armazenamento para buscar, inserir e alterar dados na aplicação. Também será explicado como funciona o esquema de validação da API, e a partir de quais classes e métodos essa camada é acessada. Esse entendimento é importante, visto que quaisquer novas interfaces que queiram acessar a API deverão fazer uso deste acesso para tal.

Enfim, a camada de apresentação será explicada (seção 4.c.), mostrando as classes responsáveis por atender as requisições feitas a partir da interface web e as páginas que geram essas requisições. Esta seção deixará mais claro como é feito o acesso às camadas responsáveis pela validação e armazenamento das entidades que compõe o Logtell, e como as mesmas podem ser manipuladas em uma camada de interface.

a. Camada de armazenamento

A camada de armazenamento é em grande parte composta de entidades, que são as classes responsáveis pelo mapeamento objeto-relacional da API. Além destas

entidades, a camada de armazenamento também conta com classes responsáveis pelo acesso programático à base de dados do Logtell.

As classes que são responsáveis pelo mapeamento objeto-relacional estão todas agrupadas no pacote de entidades da API (logteller.entity), e estão representadas no diagrama de classes da Figura 4:

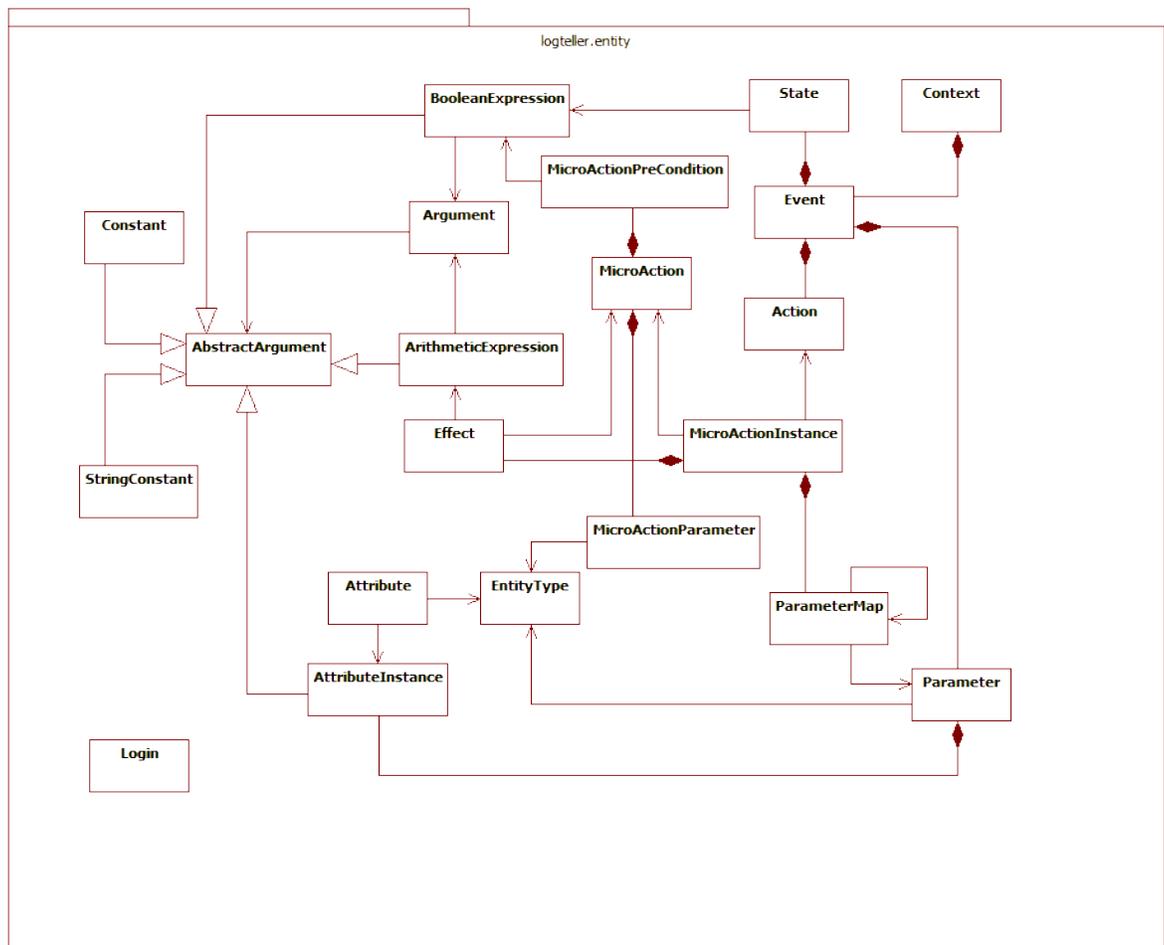


Figura 4: Diagrama de classes responsáveis pelo mapeamento objeto-relacional.

Estas classes são, em geral, entidades que representam tabelas existentes no banco de dados. O mapeamento é feito através do uso de JPA annotations, que são

anotações presentes nas classes de entidade do mapeamento objeto-relacional utilizadas para configurar esse mapeamento.

As classes representadas no diagrama anterior mapeiam as tabelas do banco de dados conforme detalhado no Apêndice 1.

As classes responsáveis pelo acesso programático ao banco de dados encontram-se agrupadas no pacote de classes de acesso ao banco de dados (logteller.dao), e possuem o sufixo DAO (Data Access Objects). Estas classes são, na verdade, EJBs (Enterprise Java Beans) que encapsulam o acesso ao banco de dados para busca, inserção, alteração e remoção das entidades mencionadas anteriormente.

Dentro destas classes, o acesso ao banco de dados é feito através da JPA (Java Persistence API), que é responsável pelo mapeamento objeto-relacional das entidades. O acesso à JPA é feito dentro das classes DAO a partir do uso de métodos específicos da JPA, ou através de JPAQL (JPA Query Language) – uma notação similar à SQL utilizada para acessar entidades mapeadas.

Por exemplo, a JPAQL da Figura 5 é utilizada dentro da classe EventDAO para buscar um evento:

```
select e
from Event e
left join fetch e.parameters p
left join fetch p.attributeInstances ai
left join fetch e.states s
left join fetch e.actions a
where e.id = :id
```

Figura 5: Exemplo de JPAQL para consulta de um objeto da classe Evento.

A consulta exibida na Figura 5 é utilizada para buscar um objeto da classe “Event”, conforme mapeado anteriormente, e para ao mesmo tempo retornar os conjuntos de objetos pertencentes ao evento – “parameters”, “states”, e “actions”.

A palavra-chave “fetch” em JPAQL é utilizada para buscar objetos de outras entidades que tenham relacionamento com uma entidade sendo buscada. Neste caso, o conjunto “attributeInstances”, pertencente à classe “Parameter”, também está sendo buscado para construir o objeto evento.

No caso dos objetos da classe “Action”, nenhuma de suas coleções é buscada nesta consulta. Sendo assim, se houver tentativa de acesso a algum destes conjuntos depois que o objeto desta consulta for retornado pelo método da classe EventDAO, ocorrerá uma exceção do tipo “LazyInitializationException”.

A partir destas definições, a seguir são listados os EJBs utilizados para acesso ao banco de dados. Por uma questão de organização, as diferentes classes do domínio da API estão espalhadas por estes métodos levando em consideração sua funcionalidade.

- **LoginDAO:** EJB utilizado para buscar objetos diretamente relacionados à classe Login. Os métodos disponibilizados por este EJB são:
 - **findByUsername:** Recebe como parâmetro um objeto da classe Login. Utiliza o atributo username deste parâmetro para buscar no banco de dados um Login com este nome.
 - **saveLogin:** Recebe como parâmetro um objeto da classe Login, e o insere no banco de dados se o mesmo não possuir um identificador. Caso o mesmo possua um identificador, o objeto com este identificador é alterado com os dados informados.

- **EntityTypeDAO:** EJB utilizado para buscar objetos diretamente relacionados à classe EntityType. Os métodos disponibilizados por este EJB são:
 - **findByName:** Recebe como parâmetro um objeto do tipo String. Utiliza este objeto para buscar uma lista de objetos da classe EntityType que possuam nome iniciando com o valor passado por parâmetro.

- **getEntityType**: Recebe como parâmetro um objeto da classe EntityType. Utiliza o identificador deste objeto para buscar no banco de dados um objeto da classe EntityType com o mesmo identificador.
 - **saveEntityType**: Recebe como parâmetro um objeto da classe EntityType, e o insere no banco de dados se o mesmo não possuir um identificador. Caso o mesmo possua um identificador, o objeto com este identificador é alterado com os dados informados.
- **AttributeDAO**: EJB utilizado para buscar objetos diretamente relacionados à classe Attribute. Os métodos disponibilizados por este EJB são:
 - **findByName**: Recebe como parâmetro um objeto do tipo String. Utiliza este objeto para buscar uma lista de objetos da classe Attribute que possuam nome iniciando com o valor passado como parâmetro.
 - **getAttribute**: Recebe como parâmetro um objeto da classe Attribute. Utiliza o identificador deste objeto para buscar no banco de dados um objeto da classe Attribute com o mesmo identificador.
 - **saveAttribute**: Recebe como parâmetro um objeto da classe Attribute, e o insere no banco de dados se o mesmo não possuir um identificador. Caso o mesmo possua um identificador, o objeto com este identificador é alterado com os dados informados.
- **MicroActionDAO**: EJB utilizado para buscar objetos diretamente relacionados à classe MicroAction. Os métodos disponibilizados por este EJB são:
 - **findByName**: Recebe como parâmetro um objeto do tipo String. Utiliza este objeto para buscar uma lista de objetos da classe MicroAction que possuam nome iniciando com o valor passado por parâmetro. As coleções contidas no objeto MicroAction não são

buscadas a partir deste método, para evitar consultas demasiadamente pesadas no banco de dados.

- **getMicroAction:** Recebe como parâmetro um objeto da classe `MicroAction`. Utiliza o identificador deste objeto para buscar no banco de dados um objeto da classe `MicroAction` com o mesmo identificador. Neste caso, as coleções contidas nesse objeto também são buscadas.
 - **saveMicroAction:** Recebe como parâmetro um objeto da classe `MicroAction`, e o insere no banco de dados se o mesmo não possuir um identificador. Os objetos das coleções (`MicroActionParameter` e `MicroActionPreCondition`) são inseridos na base de dados caso não possuam identificador, porém não podem ser atualizados. Para desassociar um `MicroActionParameter` ou um `MicroActionPreCondition`, os mesmos devem ser removidos do banco de dados a partir dos métodos de remoção disponibilizados.
 - **removeMicroActionParameter:** Recebe como parâmetro um objeto da classe `MicroActionParameter`. Remove do banco de dados um objeto que tenha o mesmo identificador.
 - **removeMicroActionPreCondition:** Recebe como parâmetro um objeto da classe `MicroActionPreCondition`. Remove do banco de dados um objeto que tenha o mesmo identificador.
- **AbstractArgumentDAO:** EJB utilizado para buscar objetos diretamente relacionados a classes que estendem `AbstractArgument`. Os métodos disponibilizados por este EJB são:
 - **findBooleanExpressionById:** Recebe como parâmetro um objeto do tipo `Long`. Utiliza este objeto para buscar no banco de dados um objeto da classe `BooleanExpression` cujo identificador seja igual ao parâmetro passado.

- **findStringConstantById:** Recebe como parâmetro um objeto da to tipo Long. Utiliza este objeto para buscar no banco de dados um objeto da classe StringConstant cujo identificador seja igual ao parâmetro passado.
- **findAttributeInstanceById:** Recebe como parâmetro um objeto da to tipo Long. Utiliza este objeto para buscar no banco de dados um objeto da classe AttributeInstance cujo identificador seja igual ao parâmetro passado.
- **findArithmeticExpressionById:** Recebe como parâmetro um objeto da to tipo Long. Utiliza este objeto para buscar no banco de dados um objeto da classe ArithmeticExpression cujo identificador seja igual ao parâmetro passado.
- **findArgument:** Recebe como parâmetros um objeto do tipo String e outro do tipo Long. A partir destes parâmetros, busca no banco de dados um objeto da classe Argument cujo tipo seja igual ao primeiro parâmetro e o valor de referência igual ao segundo.
- **saveBooleanExpression:** Recebe como parâmetro um objeto da classe BooleanExpression, e o insere no banco de dados se o mesmo não possuir um identificador. Caso o mesmo possua um identificador, o objeto correspondente no banco de dados é atualizado. Os objetos da classe Argument pertencentes ao objeto BooleanExpression também são inseridos na base de dados, se necessário. Contudo, os objetos da classe Argument devem ser explicitamente removidos da base quando forem desassociados de sua expressão booleana.
- **saveArithmeticExpression:** Recebe como parâmetro um objeto da classe ArithmeticExpression, e o insere no banco de dados se o mesmo não possuir um identificador. Caso o mesmo possua um identificador, o objeto correspondente no banco de dados é atualizado. Os objetos da classe Argument pertencentes ao objeto ArithmeticExpression também são inseridos na base de dados, se

necessário. Contudo, os objetos da classe `Argument` devem ser explicitamente removidos da base quando forem desassociados de sua expressão booleana.

- **saveStringConstant**: Recebe como parâmetro um objeto da classe `StringConstant`, e o insere no banco de dados se o mesmo não possuir um identificador. Caso o mesmo possua um identificador, o objeto correspondente no banco de dados é atualizado.
 - **removeArgument**: Recebe um objeto da classe `Argument` como parâmetro, e remove um objeto com o mesmo identificador do banco de dados. Este objeto não pode estar associado a uma expressão booleana ou aritmética para que possa ser removido.
- **ActionDAO**: EJB utilizado para buscar objetos diretamente relacionados à classe `Action`. Os métodos disponibilizados por este EJB são:
 - **getActionById**: Recebe como parâmetro um objeto da classe `Action`. Utiliza o identificador deste objeto para buscar no banco de dados um objeto da classe `Action` com o mesmo identificador. Também são buscados os objetos dos tipos `MicroActionInstance`, `ParameterMap` e `Effect` derivados desta ação.
 - **saveAction**: Recebe como parâmetro um objeto da classe `Action`, e o insere no banco de dados se o mesmo não possuir um identificador. Caso o mesmo possua um identificador, o objeto com este identificador é alterado com os dados informados. Os objetos da classe `MicroActionInstance` contidos em `Action` são inseridos no banco de dados caso não possuam identificador, porém não podem ser atualizados, e devem ser explicitamente removidos do banco de dados ao serem desassociados de sua ação. A mesma regra se aplica aos objetos das classes `ParameterMap` e `Effect` contidos em `MicroActionInstance`.

- **removeAction:** Recebe como parâmetro um objeto da classe Action, e remove do banco de dados um objeto com o mesmo identificador. Também são removidos todos os objetos das classes MicroActionInstance, ParameterMap, e Effect derivados desta ação.
 - **removeMicroActionInstance:** Recebe um objeto da classe MicroActionInstance, e remove do banco de dados um objeto com o mesmo identificador. Também são removidos todos os objetos das classes ParameterMap e Effect contidos nesse objeto MicroActionInstance.
 - **removeParameterMap:** Recebe um objeto da classe ParameterMap, e remove do banco de dados um objeto com o mesmo identificador.
 - **removeEffect:** Recebe um objeto da classe Effect, e remove do banco de dados um objeto com o mesmo identificador.
- **EventDAO:** EJB utilizado para buscar objetos diretamente relacionados à classe Event. Os métodos disponibilizados por este EJB são:
 - **getEvent:** Recebe como parâmetro um objeto da classe Event, e busca no banco de dados um objeto da classe Event com o mesmo identificador. Os objetos do tipo Event contidos neste contexto também são buscados. São também buscados os objetos das classes Parameter, Event e Action contidos nesse evento, bem como os objetos do tipo AttributeInstance contidos em Parameter.
 - **findAttributeInstance:** Recebe como parâmetros dois objetos do tipo Long, representando respectivamente os identificadores de objetos das classes Parameter e Attribute ao qual o objeto AttributeInstance a ser buscado está associado, e busca um objeto do tipo AttributeInstance no banco de dados a partir destes valores.
 - **saveEvent:** Recebe como parâmetro um objeto da classe Event, e o insere no banco de dados se o mesmo não possuir um identificador.

Caso o mesmo possua um identificador, o objeto com este identificador é alterado com os dados informados. Não salva os objetos das classes Action, State e Parameter contidos nesse contexto.

- **saveState:** Recebe como parâmetro um objeto da classe State, e o insere no banco de dados se o mesmo não possuir um identificador. Caso o mesmo possua um identificador, o objeto com este identificador é alterado com os dados informados. Não salva o objeto da classe BooleanExpression associado ao estado, considerando que este deva ser criado antes da associação.
- **saveParameter:** Recebe como parâmetro um objeto da classe Parameter, e o insere no banco de dados se o mesmo não possuir um identificador. Caso o mesmo possua um identificador, o objeto com este identificador é alterado com os dados informados. Salva os objetos da classe AttributeInstance contidos em Parameter se necessário. Contudo, os objetos da classe AttributeInstance precisam ser explicitamente removidos do banco de dados quando forem desassociados de um objeto da classe Parameter.
- **removeAttributeInstance:** Recebe como parâmetro um objeto da classe AttributeInstance, e remove do banco de dados um objeto com o mesmo identificador.
- **removeParameter:** Recebe como parâmetro um objeto da classe Parameter e remove do banco de dados um objeto com o mesmo identificador. Remove também todos os objetos da classe AttributeInstance contidos no mesmo. Para que um objeto da classe Parameter seja removido com sucesso, ele não pode ser referenciado em quaisquer objetos da classe ParameterMap.
- **removeState:** Recebe como parâmetro um objeto da classe State e remove do banco de dados um objeto com o mesmo identificador.
- **removeEvent:** Recebe como parâmetro um objeto da classe Event e remove do banco de dados um objeto com o mesmo identificador.

Remove em cascata todos os objetos das classes State, Action e Parameter contidos neste evento.

- **ContextDAO:** EJB utilizado para buscar objetos diretamente relacionados à classe Context. Os métodos disponibilizados por este EJB são:
 - **findByName:** Recebe como parâmetro um objeto do tipo String. Utiliza este objeto para buscar uma lista de objetos da classe Context cujos nomes sejam iniciados com o valor passado por Parâmetro. Os objetos do tipo Event contidos neste contexto não são buscados.
 - **getAll:** Busca uma lista contendo todos os objetos do tipo Context encontrados no banco de dados. Os objetos do tipo Event contidos neste contexto não são buscados.
 - **getContext:** Recebe como parâmetro um objeto do tipo Context, e busca no banco de dados um objeto do tipo Context com o mesmo identificador. Os objetos do tipo Event contidos neste contexto também são buscados.
 - **saveContext:** Recebe como parâmetro um objeto da classe Context, e o insere no banco de dados se o mesmo não possuir um identificador. Caso o mesmo possua um identificador, o objeto com este identificador é alterado com os dados informados. Não salva os objetos da classe Event contidos nesse contexto.
 - **removeContext:** Recebe como parâmetro um objeto da classe Context, e remove um objeto com o mesmo identificador do banco de dados. Invoca o método de remoção de eventos para cada objeto do tipo Event contido neste contexto.

É importante frisar que os métodos das classes DAO lidam exclusivamente com acesso ao banco de dados, e não realizam validações sobre a coerência dos objetos sendo passados para o banco, estando estas validações implementadas na camada de

processamento. Os dados manipulados nestas classes ainda precisam respeitar as restrições do banco de dados, ocorrendo exceções caso estas restrições não sejam respeitadas.

De todo modo, esses métodos não devem ser acessados diretamente por quaisquer interfaces que venham a fazer uso da API para edição de contextos do Logtell.

b. Camada de processamento

A camada de processamento é responsável por fazer a ligação entre a camada de apresentação, e a camada de armazenamento. Ela é composta por classes responsáveis por manter o controle transacional e por fazer as devidas validações sobre os objetos a serem inseridos no banco de dados.

O controle transacional é feito por classes do tipo EJB (Enterprise Java Bean). Estas classes estão no pacote `logteller.service`, e são divididas entre suas interfaces locais, e suas implementações localizadas no pacote `logteller.service.impl`. Estas classes são em geral chamadas pela camada de apresentação para realizar os serviços disponibilizados.

As classes responsáveis pela ligação da camada de apresentação com a camada de armazenamento estão listadas a seguir:

- **AbstractArgumentService**: Responsável por acessar a camada de armazenamento e fazer validações sobre objetos diretamente relacionados à classe `AbstractArgument`. Os métodos disponibilizados pela API estão listados a seguir:
 - **findBooleanExpressionById**;
 - **findStringConstantById**;
 - **findAttributeInstanceById**;
 - **findArithmeticExpressionById**;
 - **saveBooleanExpression**;

- **saveStringConstant;**
- **saveArithmeticExpression.**

- **ActionService:** Responsável por acessar a camada de armazenamento e fazer validações sobre objetos diretamente relacionados à classe Action. Os métodos disponibilizados pela API estão listados a seguir:
 - **findActionById;**
 - **removeAction.**

- **ContextService:** Responsável por acessar a camada de armazenamento e fazer validações sobre objetos diretamente relacionados à classe Context. Os métodos disponibilizados pela API estão listados a seguir:
 - **findContextById;**
 - **findContextByName;**
 - **removeContext;**
 - **saveContext.**

- **EventService:** Responsável por acessar a camada de armazenamento e fazer validações sobre objetos diretamente relacionados à classe Event. Os métodos disponibilizados pela API estão listados a seguir:
 - **findEventById;**
 - **findAttributeInstance;**
 - **removeEvent;**
 - **removeParameter;**
 - **removeState;**
 - **saveEvent;**
 - **saveParameter;**
 - **saveState.**

- **LoginService:** Responsável por acessar a camada de armazenamento e fazer validações sobre objetos diretamente relacionados à classe Login. Os métodos disponibilizados pela API estão listados a seguir:
 - **findLoginByUsername.**
- **MicroActionService:** Responsável por acessar a camada de armazenamento e fazer validações sobre objetos diretamente relacionados à classe MicroAction. Os métodos disponibilizados pela API estão listados a seguir:
 - **findMicroAction;**
 - **saveMicroAction.**

As classes responsáveis pela validação de dados ficam no pacote `logteller.validation`. A validação é feita a partir de uma estrutura de corrente de validação, onde regras de validação para uma determinada classe de objeto são adicionadas a uma lista, e cada uma destas regras é aplicada sobre o objeto validado, até que alguma regra falhe (retornando uma mensagem de erro), ou até que não existam mais regras para serem aplicadas.

Como o autômato que define um contexto é uma estrutura de grande complexidade e que possui características específicas que exigiriam validações igualmente complexas, fugiria ao escopo deste trabalho uma tentativa de mapear todas as possíveis situações que exigiriam validação para os elementos que compõem um contexto.

Sendo assim, foram criadas algumas validações para exemplificar o funcionamento da corrente de validação. A estrutura da corrente de validação tem como vantagem centralizar a adição de regras de validação em uma classe específica para cada tipo de objeto, tornando mais fácil a criação de novas regras.

A estrutura básica da corrente de validação é baseada em um conjunto de classes:

- **Rule:** Localizada no pacote `logteller.validation`, trata-se de uma interface que define o funcionamento básico de uma regra. Define três métodos a serem implementados por classes que herdam esta interface:
 - **getErrorMessage:** Define a mensagem de erro a ser retornada quando a regra falha.
 - **setValidationObject:** Utilizado para definir o objeto sobre o qual a regra de validação será aplicada.
 - **validate:** Método que contém a lógica da regra de validação.

- **AbstractChain:** Localizada no pacote `logteller.validation`, trata-se de uma classe abstrata que define o funcionamento básico de uma corrente de validação. Contém uma lista encadeada de objetos que implementam a interface `Rule`. Contém os seguintes métodos:
 - **getErrorMessage:** Retorna a `String` contendo a mensagem de erro contida nessa corrente de validação após a execução do método `doChain`.
 - **addRule:** Recebe como parâmetro um objeto do tipo `Rule`, e adiciona esse objeto à lista de regras de validação.
 - **removeRule:** Recebe como parâmetro um objeto do tipo `Rule`, e remove esse objeto da lista de regras de validação.
 - **clearChain:** Limpa a lista de regras de validação, removendo todas as regras anteriormente adicionadas.
 - **doChain:** Método abstrato que deve ser implementado pelas subclasses de `AbstractChain`. Deve executar as regras contidas na lista de regras de validação.

- **StoppableChain:** Localizada no pacote `logteller.validation.chains`, trata-se de uma implementação de `AbstractChain`, que executa a corrente de validação até encontrar um erro, ou até exaurir as regras de validação. Implementa o seguinte método:
 - **doChain:** Retorna falso quando alguma das regras na lista falha na sua validação. Neste caso, a mensagem de erro definida em `AbstractChain` recebe a mensagem de erro definida no objeto `Rule` cuja validação falhou.

- **ChainBuilder:** Interface genérica que define o funcionamento de classes que construam objetos do tipo `AbstractChain`. Essa interface define o seguinte método:
 - **build:** Recebe como parâmetro um objeto de uma classe que estenda `Object`, e retorna um objeto de uma classe que estenda `AbstractChain`. Espera-se que construa uma corrente de validação com Regras que validem o objeto passado por parâmetro.

O exemplo criado para exemplificar o funcionamento da corrente de validação encontra-se no pacote `logteller.action.microaction`, e contém as seguintes classes:

- **MicroActionDurationRule:** Implementação de `Rule` onde a implementação do método `validate` verifica se a duração da micro ação não é nula e é maior que zero.
- **MicroActionNameRule:** Implementação de `Rule` onde a implementação do método `validate` verifica se o nome da micro ação não é nula e é diferente de uma `String` vazia.
- **MicroActionChainBuilder:** Implementação de `ChainBuilder` cuja implementação de `build` recebe um objeto do tipo `MicroAction` como

parâmetro e retorna uma `StoppableChain` contendo as regras `MicroActionNameRule` e `MicroActionDurationRule` em sua lista.

Muitas outras regras de validação poderiam ser criadas para uma micro ação. Para que as mesmas fossem aplicadas, bastaria a criação de uma nova implementação de `Rule` e que esta implementação fosse adicionada a lista de regras em `MicroActionChainBuilder`.

c. Camada de apresentação

A camada de apresentação é apenas uma forma de exemplificar como os métodos disponibilizados pela camada de processamento da API podem ser acessados por uma interface gráfica para buscar – e possivelmente inserir – dados no modelo do Logtell.

A interface de exemplo é composta pelas classes responsáveis por atender as requisições originadas das páginas onde ocorre a exibição dos dados e a interação com o usuário. Estas classes, por sua vez, acessam classes da camada de processamento para buscar ou inserir os dados necessários para atender as requisições. As classes da camada de apresentação, idealmente, devem apenas lidar com problemas relativos à exibição e entrada de dados, uma vez que as camadas inferiores são responsáveis pela validação e armazenamento dos mesmos.

As páginas XHTML acessam os métodos das classes da camada de apresentação a partir do uso de expressões de linguagem do JSF (framework utilizado no XHTML para construir as páginas) e do conceito de injeção de dependência do JBoss Seam. A Figura 6 mostra um exemplo do funcionamento das páginas e como as mesmas se relacionam com as classes:

```

<ui:define name="body">
  <h:form id="contexts_form">
    <div id="context_search_errors">
      <h:messages />
    </div>

    <div id="context_search_area">
      <div id="context_search_components">
        <div id="context_search_field">
          <p><h:inputText value="#{contextsBean.contextSearch}"
            styleClass="context_search_field" /></p>
        </div>

        <div id="context_search_button">
          <p><h:commandButton action="#{contextsBean.findContexts}"
            value="#{messages['btn.find']}"
            styleClass="context_search_button" /></p>
        </div>
      </div>
    </div>
  </div>
</div>

```

Figura 6: Exemplo de utilização de expressões de linguagem em página XHTML.

O trecho de código acima mostra como uma página XHTML utiliza expressões de linguagem do JSF para acessar métodos disponibilizados pelas classes da camada de apresentação. Na quarta linha, a tag `<h:inputText>` será renderizada pelo JSF como uma tag `<input type="text">`. O valor deste input será passado para a classe `ContextsBean` através do método `setContextSearch(String)`, que pode ser observado na Figura 7:

```

@Name(value = "contextsBean")
@Scope(ScopeType.CONVERSATION)
public class ContextsBean {

    // Attributes.

    @SuppressWarnings("unused")
    private static Logger logger = LoggerFactory.getLogger(ContextsBean.class);

    private List<Context> contextList;

    private String contextSearch;

    public String getContextSearch() {
        return contextSearch;
    }

    public void setContextSearch(String contextSearch) {
        this.contextSearch = contextSearch;
    }
}

```

Figura 7: Definição da classe ContextsBean, com o método setContextSearch(String).

A classe `logteller.bean.ContextsBean` recebe a anotação `@Name`, de forma que o JBoss Seam faz a injeção de dependência de um objeto desta classe a partir do nome “contextsBean”, que é utilizado pela expressão de linguagem de expressão na página `contexts.xhtml`. Desta forma, a expressão de linguagem do JSF, acessa o método `setContextSearch()` da classe `logteller.bean.ContextsBean` através do nome “contextsBean”.

Para o melhor entendimento das classes que compõem a camada de apresentação, primeiramente serão apresentadas as páginas XHTML responsáveis pela entrada e saída de dados. As páginas do visualizador de contextos do Logtell estão listadas a seguir:

- `action_detail.xhtml`:

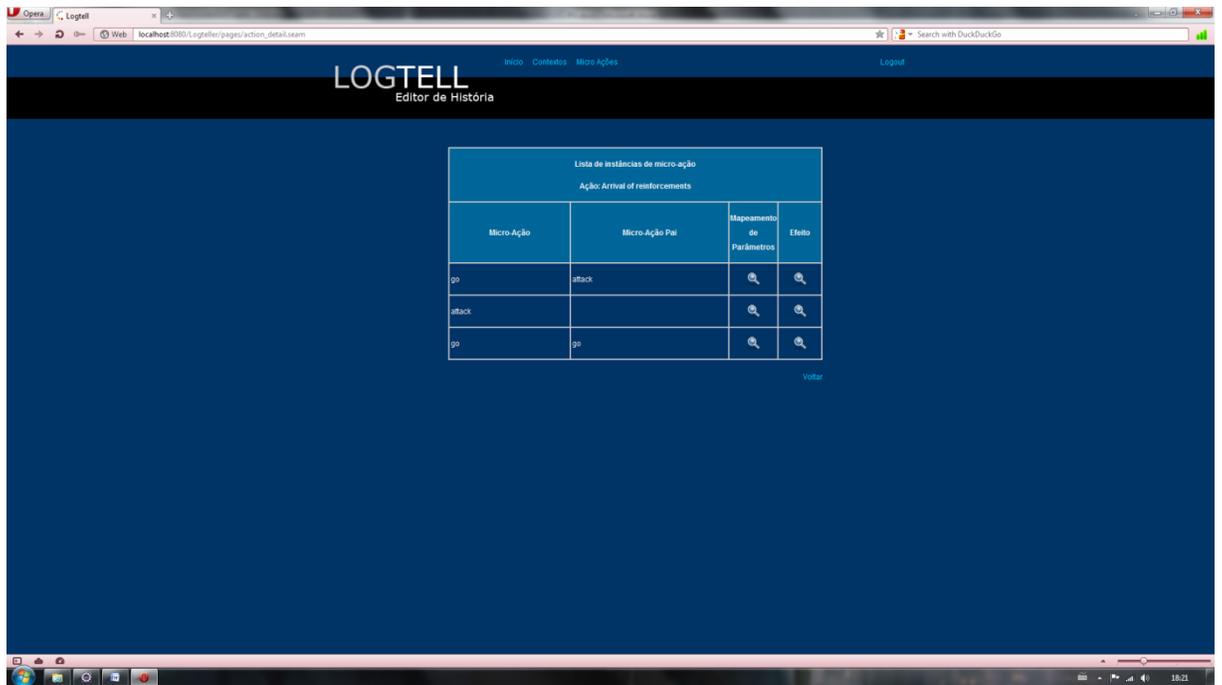


Figura 8: Captura de tela referente à página action_detail.xhtml.

Responsável por detalhar uma ação, a partir da lista de instâncias de micro-ação que compõe a ação. Para cada instância de micro-ação listada, pode ser exibido o mapeamento de parâmetros ou os efeitos da execução dessa instância de micro-ação.

As requisições relativas a esta tela são atendidas pela classe `logteller.bean.ActionDetailBean`. Esta classe possui métodos para atender às requisições da tela para buscar um evento com todos os seus atributos preenchidos, bem como sua lista de efeitos e o mapeamento de parâmetros.

- `attribute_popup_view.xhtml`



Figura 9: Captura de tela referente à página attribute_popup_view.xhtml.

Responsável por exibir a lista de atributos de um parâmetro do evento. Essa página XHTML é na verdade um popup do tipo modal, exigindo que o usuário feche o popup antes que possa interagir novamente com a página pai.

As requisições relativas a esta tela são atendidas pela classe `logteller.bean.EventDetailBean`, que também atende às requisições da página XHTML `event_detail.xhtml`, que contém este modal.

- `context_detail.xhtml`

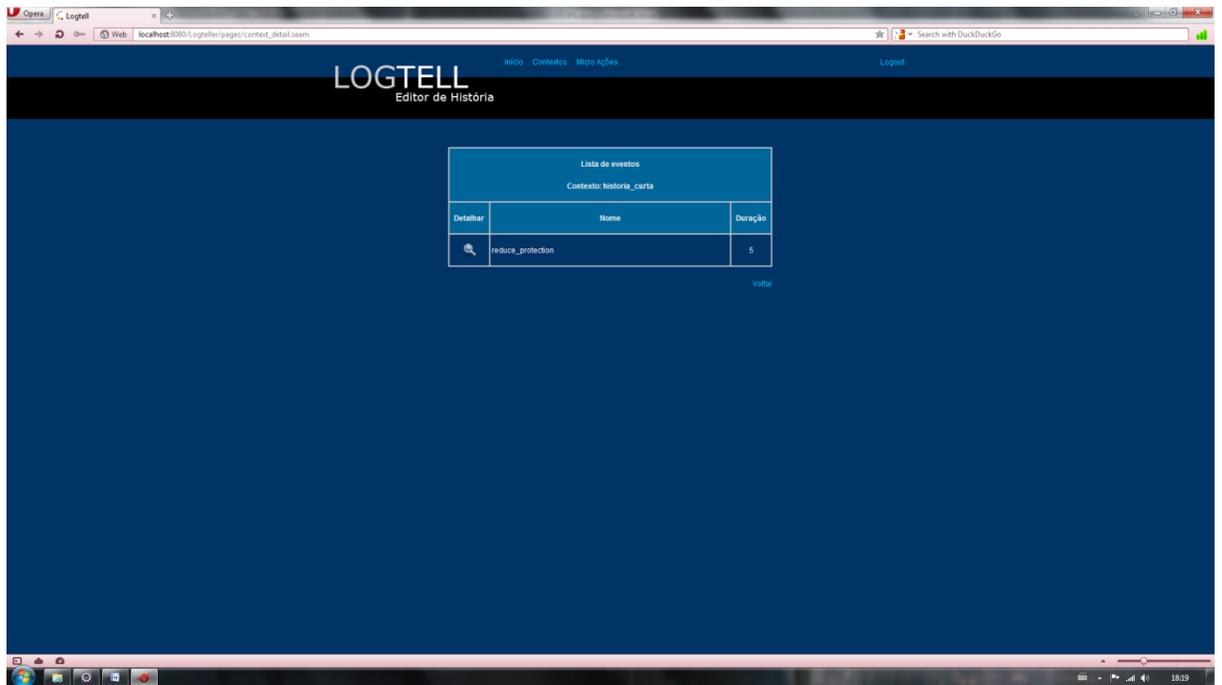


Figura 10: Captura de tela referente à página context_detail.xhtml.

Responsável por exibir a lista de eventos que compõe um contexto. Para cada evento é exibida a duração do mesmo, e a opção de detalhar o evento, exibindo suas coleções de parâmetros, estados, e ações.

As requisições relativas a esta tela são atendidas pela classe `logteller.bean.ContextDetailBean`. Esta classe possui métodos para atender às requisições da tela para buscar um contexto com todos os seus atributos preenchidos, bem como suas listas de parâmetros, estados e ações.

- `contexts.xhtml`

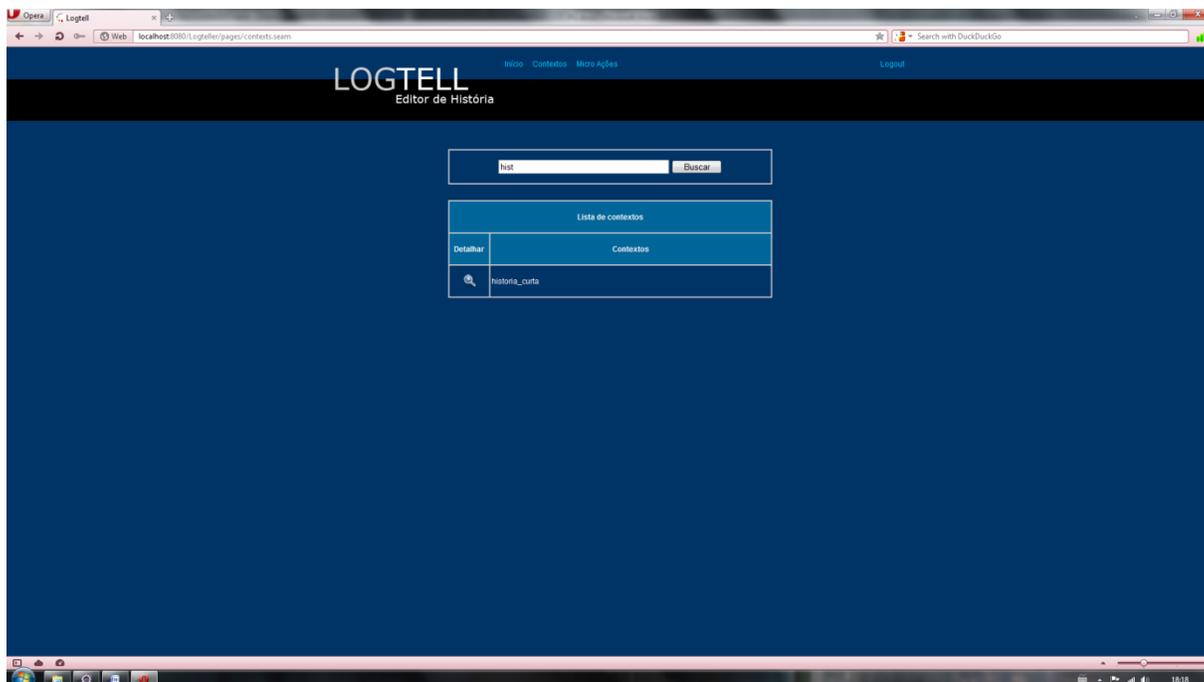


Figura 11: Captura de tela referente à página contextos.xhtml.

Responsável por buscar a lista de contextos cadastrados no banco de dados. Possui uma caixa de texto para que o contexto possa ser buscado por nome.

As requisições relativas a esta tela são atendidas pela classe `logteller.bean.ContextsBean`. Esta classe possui o método para atender às requisições da tela para encontrar um contexto a partir de seu nome.

- `effect_popup_view.xhtml`

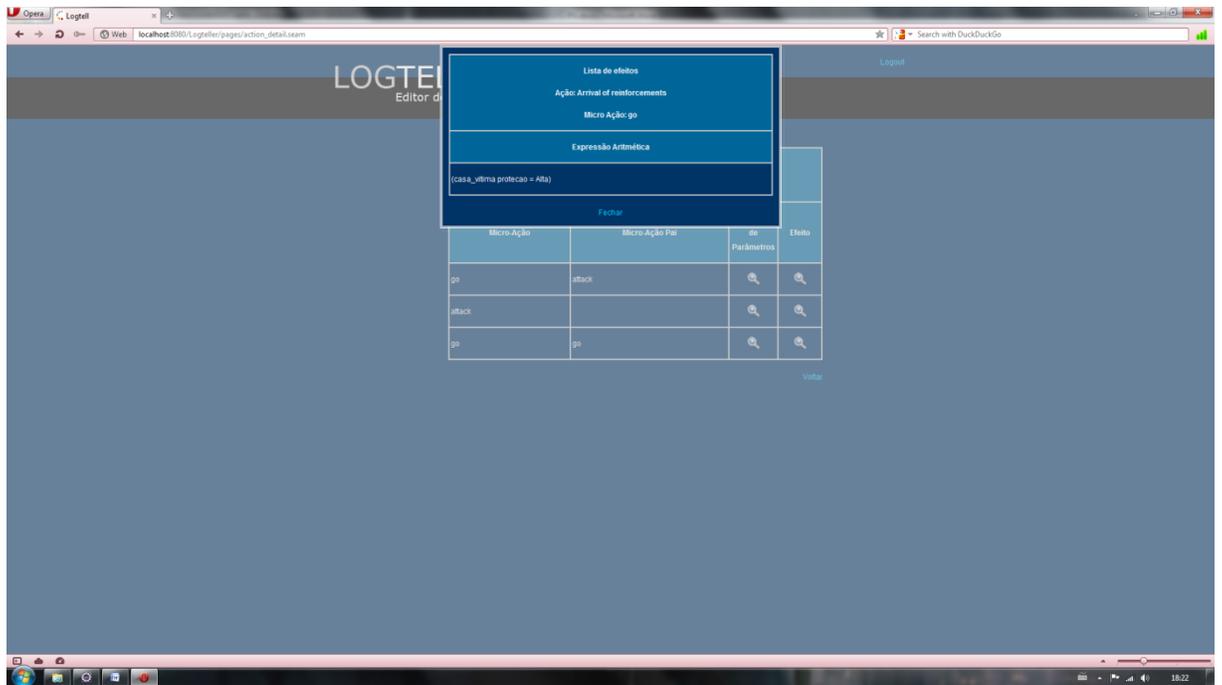


Figura 12: Captura de tela referente à página effect_popup_view.xhtml.

Responsável por exibir a lista de expressões aritméticas que definem os efeitos de uma ação. Essa página XHTML é na verdade um popup do tipo modal, exigindo que o usuário feche o popup antes que possa interagir novamente com a página pai.

As requisições relativas a esta tela são atendidas pela classe `logteller.bean.ActionDetailBean`, que também atende às requisições da página XHTML `action_detail.xhtml`, que contém este modal.

- `event_detail.xhtml`

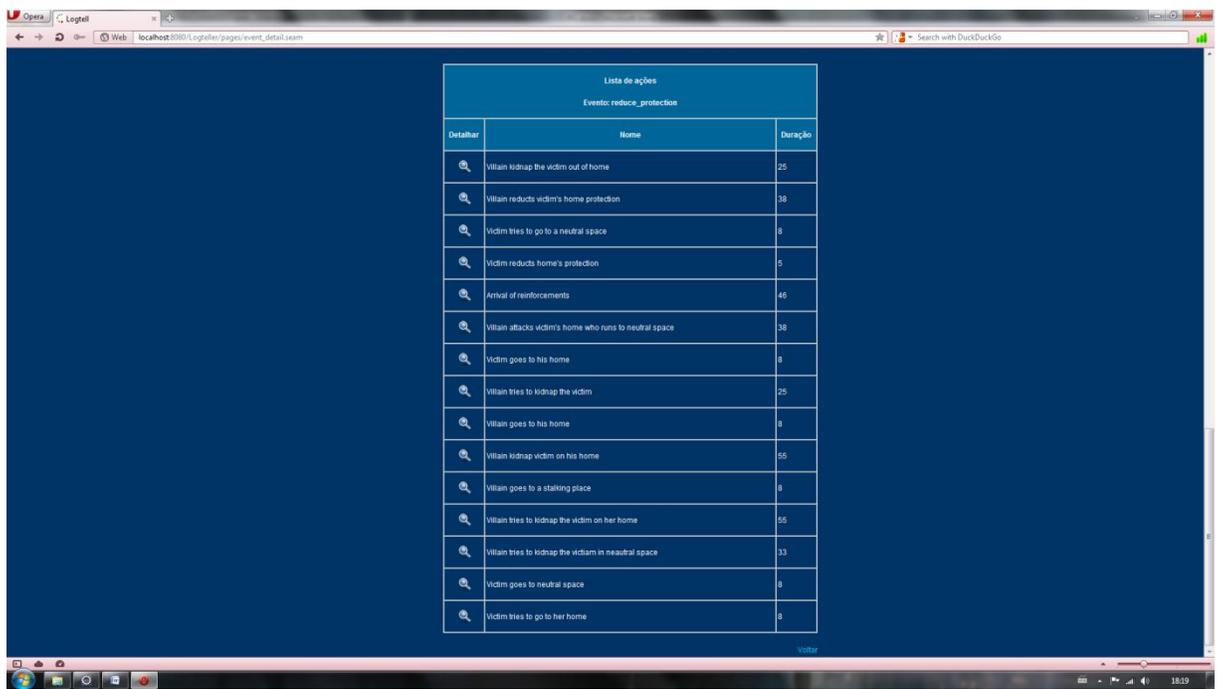
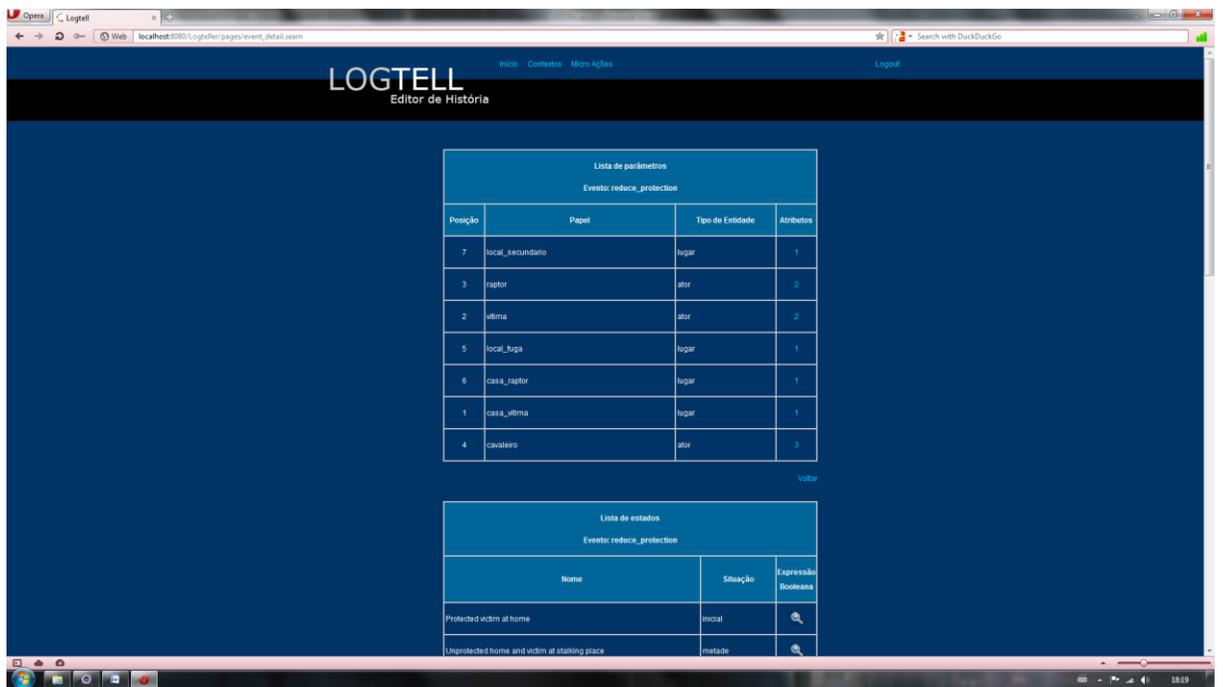


Figura 13: Capturas de tela referente à página event_detail.xhtml.

Responsável por detalhar um evento, a partir de suas listas e parâmetros, estados e ações.

Para cada parâmetro listado, é mostrada sua posição, papel, tipo de entidade e quantidade de atributos. É possível exibir estes atributos ao clicar no número que indica a quantidade de atributos.

Para cada estado listado, é mostrado seu nome, situação, e pode-se exibir um popup contendo a expressão booleana que define o estado.

Para cada ação listada, é mostrado seu nome e duração, além de um ícone que pode ser clicado para exibir a tela de detalhamento desta ação, mencionada anteriormente (action_detail.xhtml).

As requisições relativas a esta tela são atendidas pela classe `logteller.bean.EventDetailBean`. Esta classe possui métodos para atender às requisições da tela para exibir o detalhamento de um evento, com todos os seus atributos e coleções, bem como exibir a lista de atributos de um parâmetro ou buscar a expressão booleana que define o estado.

- `expression_popup_view.xhtml`

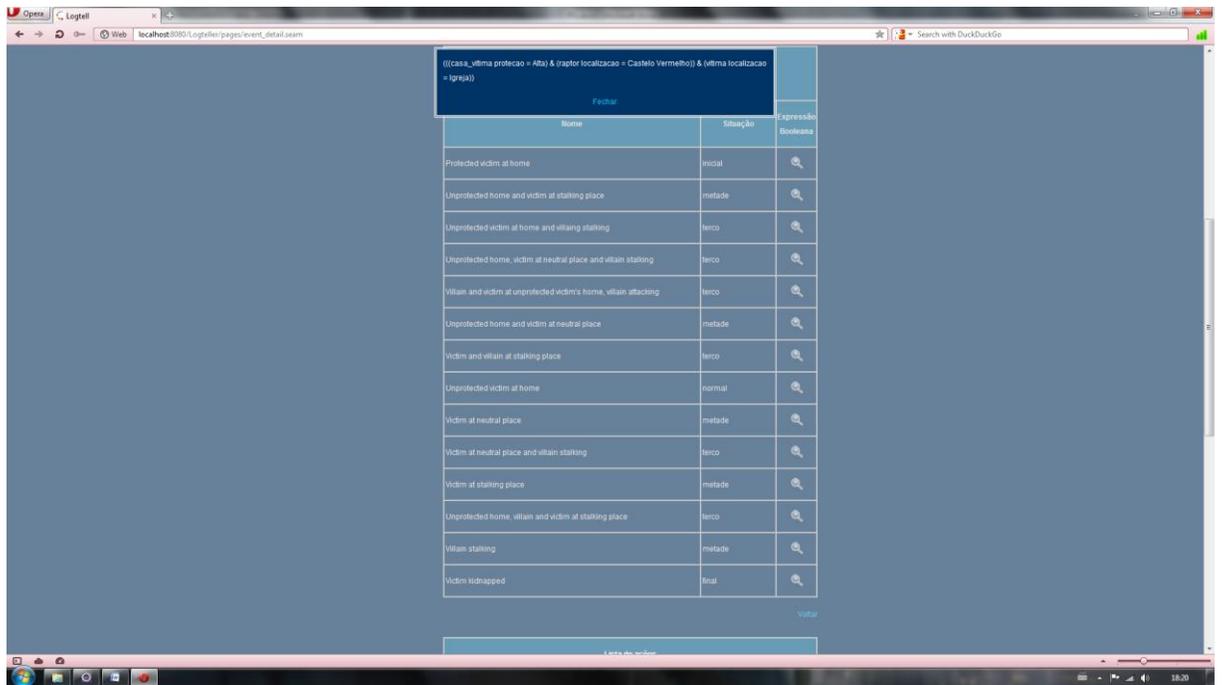


Figura 14: Capturas de tela referente à página expression_popup_view.xhtml.

Responsável por exibir a lista de expressões booleanas que definem o estado de um evento. Essa página XHTML é na verdade um popup do tipo modal, exigindo que o usuário feche o popup antes que possa interagir novamente com a página pai.

As requisições relativas a esta tela são atendidas pela classe logteller.bean.EventDetailBean, que também atende às requisições da página XHTML event_detail.xhtml, que contém este modal.

- login.xhtml

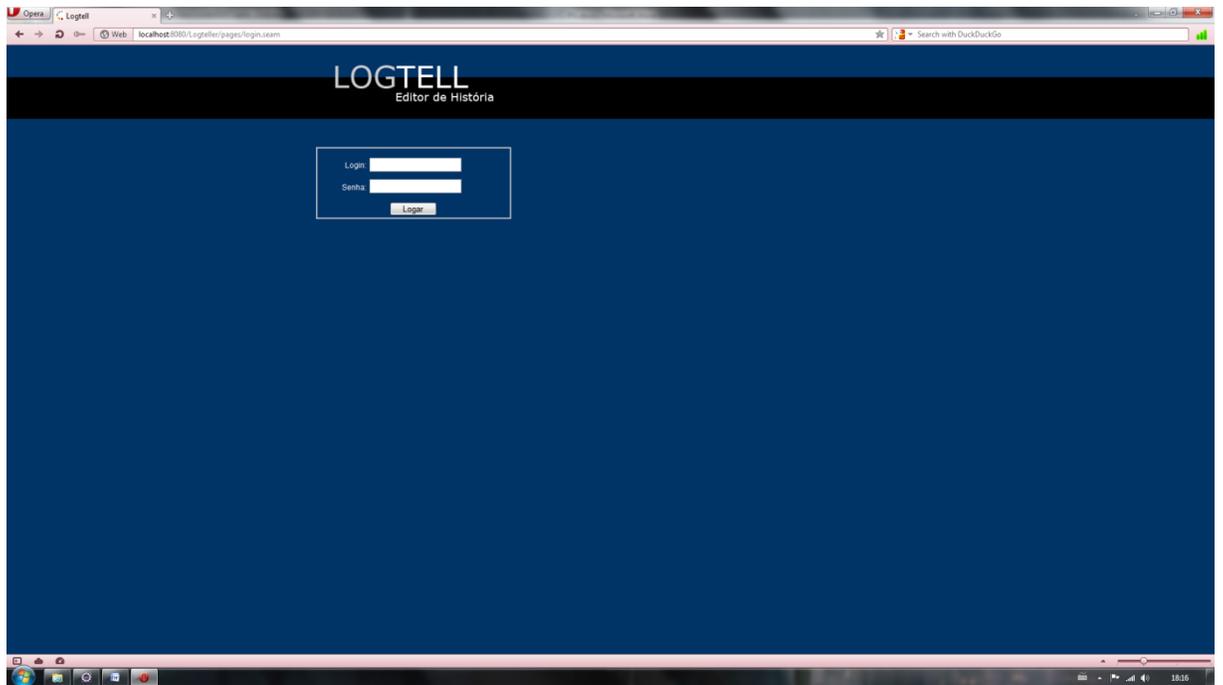


Figura 15: Capturas de tela referente à página login.xhtml.

Responsável pela funcionalidade de login da ferramenta. As requisições relativas a esta tela são atendidas pela classe `logteller.authenticator.Authenticator`, que possui métodos para verificação de nome de usuário e senha.

- `micro_actions.xhtml`

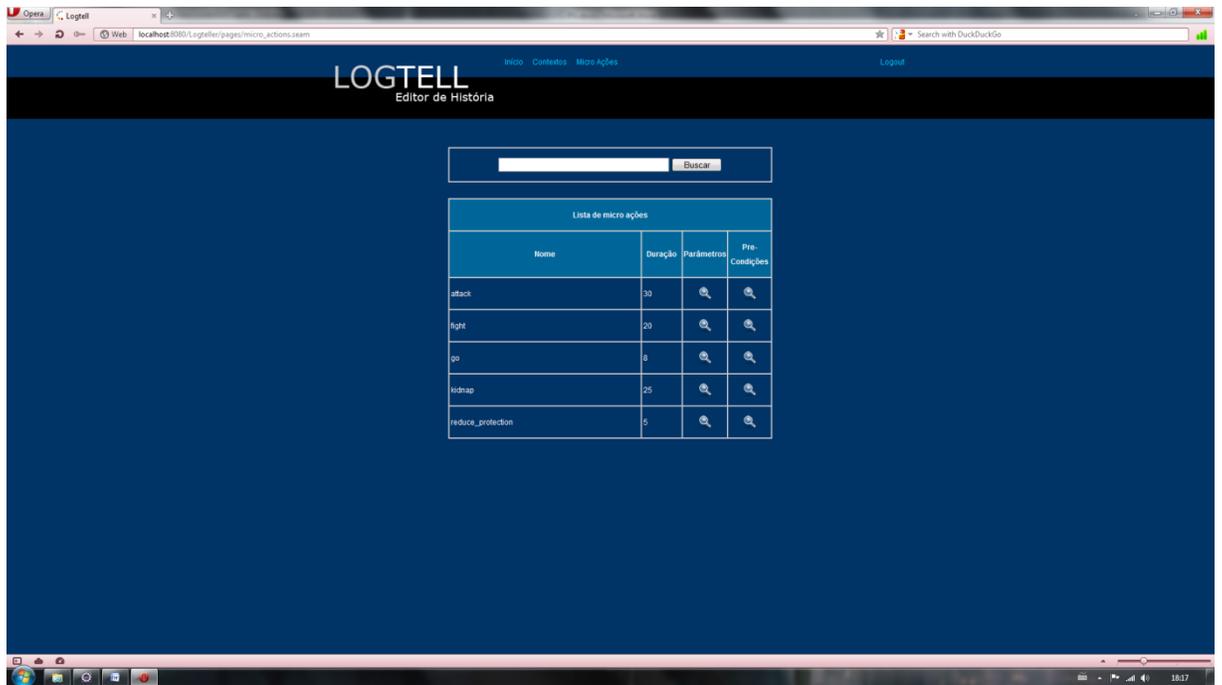


Figura 16: Capturas de tela referente à página micro_actions.xhtml.

Responsável por buscar a lista de micro ações cadastrada no banco de dados. Possui uma caixa de texto para que a micro ação possa ser buscada por nome, além de listar a duração de cada micro ação e links para visualizar as listas de parâmetros e pré-condições de cada micro ação.

As requisições relativas a esta tela são atendidas pela classe `logteller.bean.MicroActionsBean`. Esta classe possui o método para atender às requisições da tela para encontrar uma micro-ação partir de seu nome. Possui também métodos para buscar os parâmetros e pré-condições de cada micro ação.

- `microaction_parameter_popup_view.xhtml`

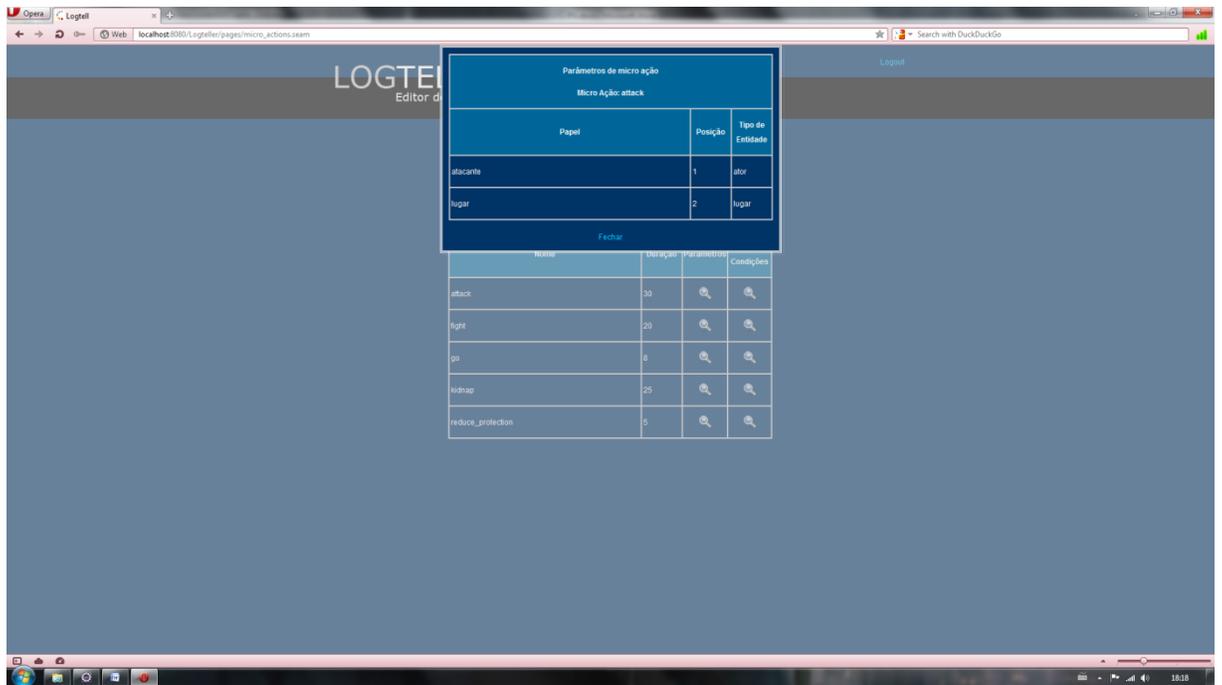


Figura 17: Capturas de tela referente à página `microaction_parameter_popup_view.xhtml`.

Responsável por exibir a lista de parâmetros de micro ação. Essa página XHTML é na verdade um popup do tipo modal, exigindo que o usuário feche o popup antes que possa interagir novamente com a página pai.

As requisições relativas a esta tela são atendidas pela classe `logteller.bean.MicroActionsBean`, que também atende às requisições da página XHTML `micro_actions.xhtml`, que contém este modal.

- `microaction_precondition_popup_view.xhtml`

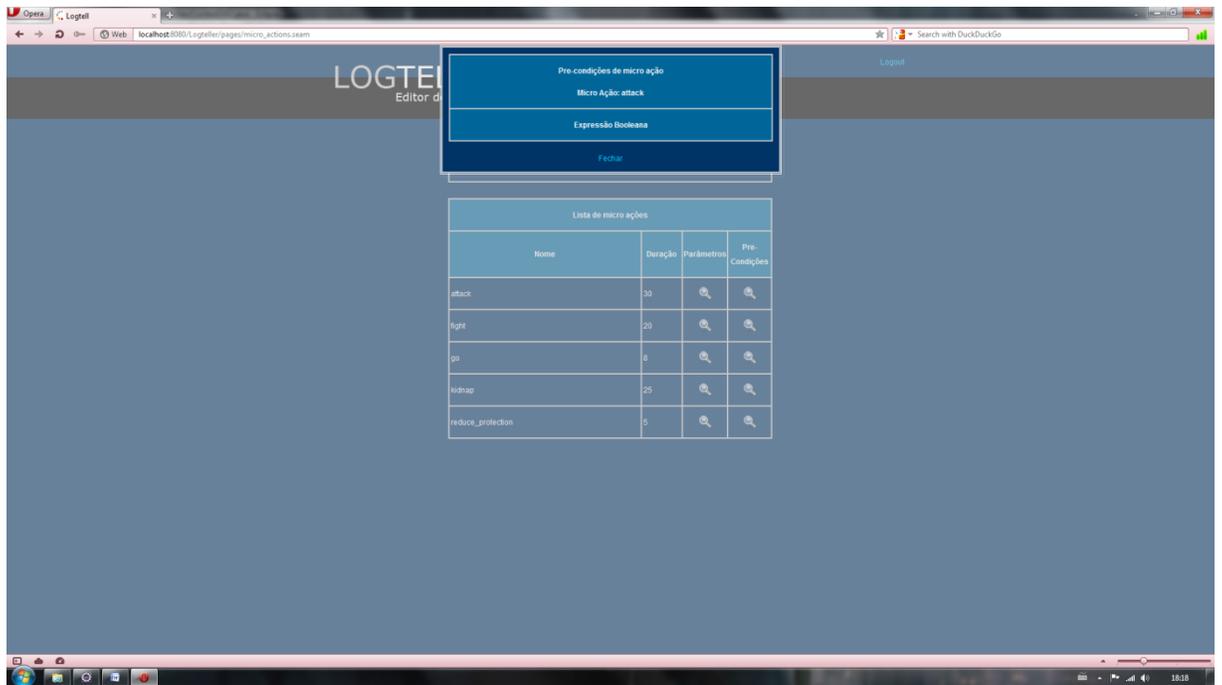


Figura 18: Capturas de tela referente à página `microaction_precondition_popup_view.xhtml`.

Responsável por exibir a lista de expressões booleanas que definem as pré-condições de micro ação. Essa página XHTML é na verdade um popup do tipo modal, exigindo que o usuário feche o popup antes que possa interagir novamente com a página pai.

As requisições relativas a esta tela são atendidas pela classe `logteller.bean.MicroActionsBean`, que também atende às requisições da página XHTML `micro_actions.xhtml`, que contém este modal.

- `parameter_map_popup_view.xhtml`

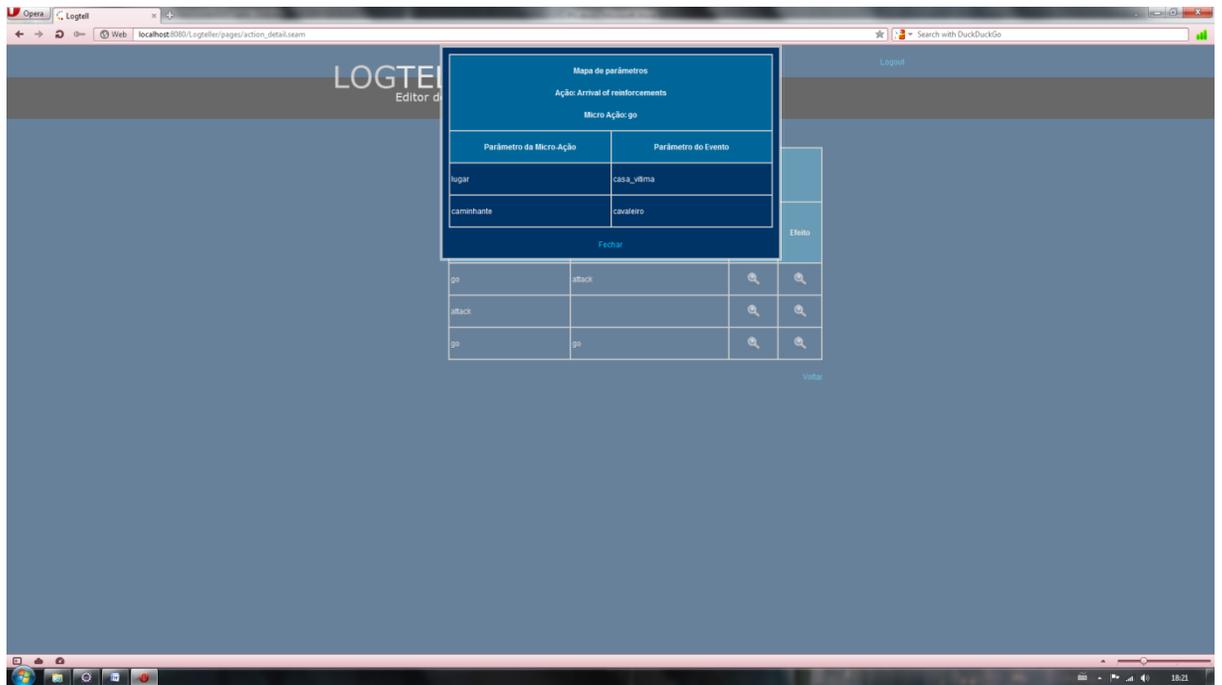


Figura 19: Capturas de tela referente à página parameter_map_popup_view.xhtml.

Responsável por exibir o mapeamento entre parâmetros de micro ação e parâmetros de evento dentro de uma instância de micro ação. Essa página XHTML é na verdade um popup do tipo modal, exigindo que o usuário feche o popup antes que possa interagir novamente com a página pai.

As requisições relativas a esta tela são atendidas pela classe `logteller.bean.ActionDetailBean`, que também atende às requisições da página XHTML `action_detail.xhtml`, que contém este modal.

5. RESULTADOS

Neste capítulo será demonstrado com exemplos como a existência da interface gráfica para visualização de contextos, construída sobre a API para manipulação de dados do Logtell torna mais prático e menos sujeito a erros a tarefa de visualizar dados do Logtell.

a. Parâmetros de micro ação

Para visualizar parâmetros de uma micro ação por meio de consultas SQL, seria necessário executar, pelo menos, as duas consultas da Figura 20:

```
select *
from micro_acao
order by id;

select ma.nome, ma.duracao_segundos, pma.papel, pma.posicao, e.nome
from micro_acao ma
  inner join
    parametro_micro_acao pma
      on ma.id = pma.id_micro_acao
  inner join
    entidade e
      on pma.id_tipo_entidade = e.id
where ma.id = 3;
```

Figura 20: Consulta SQL para buscar parâmetros de micro-ação.

A primeira consulta seria necessária para selecionar a micro ação (no caso, a micro ação “kidnap”, que possui o ID 3. Depois da escolha da micro ação pelo identificador, a segunda consulta poderia ser executada obtendo o resultado da Figura 21:

| | nome character varying | duracao_segundos numeric | papel character varying | posicao numeric | nome character varying |
|---|---------------------------|-----------------------------|----------------------------|--------------------|---------------------------|
| 1 | kidnap | 25 | raptor | 1 | ator |
| 2 | kidnap | 25 | vitima | 2 | ator |

Figura 21: Resultado da consulta SQL por parâmetros de micro-ação.

Contudo, ao fazer a navegação pela tela, bastaria selecionar a micro ação a partir da tela de busca de micro ação, representada na Figura 22:

| Lista de micro ações | | | |
|----------------------|---------|------------|---------------|
| Nome | Duração | Parâmetros | Pre-Condições |
| attack | 30 | 🔍 | 🔍 |
| fight | 20 | 🔍 | 🔍 |
| go | 8 | 🔍 | 🔍 |
| kidnap | 25 | 🔍 | 🔍 |
| reduce_protection | 5 | 🔍 | 🔍 |

Figura 22: Busca de micro-ações pela tela.

Após a seleção da micro ação, basta clicar na lupa da coluna parâmetros na linha da micro ação selecionada – no caso “kidnap” – resultando no popup da Figura 23:

| Parâmetros de micro ação | | | |
|--------------------------|---------|------------------|--|
| Micro Ação: kidnap | | | |
| Papel | Posição | Tipo de Entidade | |
| raptor | 1 | ator | |
| vitima | 2 | ator | |
| Fechar | | | |

| Nome | Duração | Parâmetros | Condições |
|--------|---------|---|---|
| attack | 30 |  |  |
| fight | 20 |  |  |
| go | 8 |  |  |
| kidnap | 25 |  |  |

Figura 23: Parâmetros da micro-ação kidnap.

b. Expressões booleanas de um estado

Para visualizar a expressão booleana de um estado por meio de consultas SQL, seria necessário executar, inicialmente, as consultas da Figura 24 para seleccionar o estado:

```

select *
from contexto c
order by c.id;

select *
from evento e
where e.id_contexto = 1
order by e.id;

select *
from estado e
where e.id_evento = 1
order by e.id;

```

Figura 24: Consultas SQL para seleção de um estado.

A partir da seleção do estado, pode-se rodar a consulta SQL da Figura 25 para buscar a expressão booleana do mesmo pelo identificador do estado selecionado (no caso, foi selecionado o estado de ID 34, cujo nome é “Victim at neutral place and villain stalking”):

```

select a1.tipo, a1.ref_valor, eb.operador, a2.tipo, a2.ref_valor
from expressao_booleana eb
  inner join
    argumento a1
      on eb.id_argumento1 = a1.id
  inner join
    argumento a2
      on eb.id_argumento2 = a2.id
where eb.id = 34;

```

Figura 25: Consultas SQL para seleção da expressão booleana de um estado.

Esta consulta tem o resultado da Figura 26:

| | tipo | ref_valor | operador | tipo | ref_valor |
|---|--------------------|-----------|------------------------|--------------------|-----------|
| | character varying | numeric | character varying(255) | character varying | numeric |
| 1 | expressao_booleana | 28 | & | expressao_booleana | 33 |

Figura 26: Resultado da consulta SQL da Figura 25.

A partir desse resultado (“Expressão Booleana 28” AND “Expressão Booleana 33”), será necessário buscar estas duas expressões booleanas também, conforme as consultas SQL da Figura 27:

```
select eb.id, a1.tipo, a1.ref_valor, eb.operador, a2.tipo, a2.ref_valor
from expressao_booleana eb
  inner join
    argumento a1
      on eb.id_argumento1 = a1.id
  inner join
    argumento a2
      on eb.id_argumento2 = a2.id
where eb.id in (28, 33);
```

Figura 27: Consultas SQL para seleção das expressões booleanas derivadas do resultado da Figura 26.

Esta consulta gera resultado da Figura 28:

| | id numeric | tipo character varying | ref_valor numeric | operador character varying(255) | tipo character varying | ref_valor numeric |
|---|---------------|---------------------------|----------------------|------------------------------------|---------------------------|----------------------|
| 1 | 28 | instancia_atributo | 2 | = | constante_string | 6 |
| 2 | 33 | expressao_booleana | 1 | & | expressao_booleana | 10 |

Figura 28: Resultado da consulta SQL da Figura 27.

A partir dos resultados obtidos até então, a expressão booleana passa a ser: (“Instancia Atributo 2” = “Constante String 6”) AND (“Expressão Booleana 1” AND “Expressão Booleana 10”). Com isso as consultas da Figura 29 precisarão ser executadas:

```

select a.nome, p.papel
from instancia_atributo ia
  inner join
    atributo a
    on ia.id_atributo = a.id
  inner join
    parametro p
    on ia.id_parametro = p.id
where ia.id = 2;

select *
from constante cs
where cs.id = 6;

select eb.id, a1.tipo, a1.ref_valor, eb.operador, a2.tipo, a2.ref_valor
from expressao_booleana eb
  inner join
    argumento a1
    on eb.id_argumento1 = a1.id
  inner join
    argumento a2
    on eb.id_argumento2 = a2.id
where eb.id in (1, 10);

```

Figura 29: Consultas SQL para seleção das entidades derivadas do resultado da Figura 28.

A partir destas consultas SQL, são gerados os resultados da Figura 30:

| | nome character varying | papel character varying |
|----------|---------------------------|----------------------------|
| 1 | localizacao | vitima |

| | id numeric | valor character varying |
|----------|---------------|----------------------------|
| 1 | 6 | Igreja |

| | id numeric | tipo character varying | ref_valor numeric | operador character varying(255) | tipo character varying | ref_valor numeric |
|----------|---------------|---------------------------|----------------------|------------------------------------|---------------------------|----------------------|
| 1 | 1 | instancia_atributo | 7 | = | constante_string | 5 |
| 2 | 10 | instancia_atributo | 4 | = | constante_string | 3 |

Figura 30: Resultados das consultas SQL da Figura 29.

A partir desses resultados a expressão montada passa a ser: (“Vitima – Localização” = “Igreja”) AND ((“Instancia Atributo 7” = “Constante String 5”) AND

(“Instancia Atributo 10” = “Constante String 3”). Com isso, as consultas SQL da Figura 31 precisam ser executadas:

```

select ia.id, a.nome, p.papel
from instancia_atributo ia
  inner join
    atributo a
    on ia.id_atributo = a.id
  inner join
    parametro p
    on ia.id_parametro = p.id
where ia.id in (4, 7);

select *
from constante cs
where cs.id in (5, 3);

```

Figura 31: Consultas SQL para seleção das entidades derivadas do resultado da Figura 30.

As consultas acima geram os resultados da Figura 32:

| | id numeric | nome character varying | papel character varying |
|----------|-----------------------------|---|--|
| 1 | 7 | protecao | casa_vitima |
| 2 | 4 | localizacao | raptor |

| | id numeric | valor character varying |
|----------|-----------------------------|--|
| 1 | 3 | Floresta |
| 2 | 5 | Alta |

Figura 32: Resultados das consultas SQL da Figura 31.

Com esses resultados, chega-se à expressão booleana completa para o evento selecionado: (“Vitima – Localização” = “Igreja”) AND ((“Casa Vitima – Proteção” = “Alta”) AND (“Raptor – Localização” = “Floresta”)).

Para visualizar a mesma expressão booleana a partir da ferramenta, é preciso inicialmente selecionar o contexto a partir da tela de seleção de contextos da Figura 33 (existe apenas um cadastrado):



Figura 33: Resultado da busca de contexto a partir da ferramenta.

Então deve-se clicar na lupa para detalhar o contexto selecionado, listando seus eventos, conforme demonstrado na Figura 34 (existe apenas um cadastrado):



Figura 34: Visualização dos eventos do contexto “historia_curta”.

Na lista de eventos, deve-se clicar novamente na lupa, para detalhar o evento selecionado exibindo sua lista de estados, parâmetros e ações, resultando na tela da Figura 35:

| Lista de estados | | |
|--|----------|---|
| Evento: reduce_protection | | |
| Nome | Situação | Expressão Booleana |
| Victim and villain at stalking place | terco |  |
| Victim at neutral place and villain stalking | terco |  |
| Villain stalking | metade |  |
| Unprotected home, villain and victim at stalking place | terco |  |

Figura 35: Visualização dos estados do evento “reduce_protection”.

Rolando a tela até a lista de estados do evento, pode-se clicar na lupa da coluna Expressão Booleana para visualizar a expressão booleana do estado. No caso exemplificado acima, trata-se do estado “Victim at neutral place and villain stalking”. Desta forma, é exibido o popup da Figura 36:

| <pre>((vitima localizacao = Igreja) & ((casa_vitima protecao = Alta) & (raptor localizacao = Floresta)))</pre> | | 2 |
|--|----------|---|
| <p>Fechar</p> | | Voltar |
| <p>Lista de estados</p> <p>Evento: reduce_protection</p> | | |
| Nome | Situação | Expressão Booleana |
| Victim and villain at stalking place | terco |  |
| Victim at neutral place and villain stalking | terco |  |

Figura 36: Visualização da expressão booleana do estado “Victim at neutral place and villain stalking”.

6. CONCLUSÃO

Através da arquitetura sugerida e da implementação da proposta para uma API de visualização e edição de contextos do Logtell, foi possível criar uma simples ferramenta que, ao fazer uso dos métodos disponibilizados da camada de processamento, exibe graficamente vários aspectos das estruturas que formam o contexto de uma história do Logtell.

Contudo, existem muitas formas de ampliar o uso desta API, através da adição de regras de validação para a inserção de contextos, e da incorporação do já existente gerador de políticas para esses contextos.

Além disso, esta API também possibilita a criação de ferramentas de interfaces completas para criação e edição de contextos de forma gráfica e visual, que facilitariam a tarefa dos autores ao escrever histórias para o Logtell sem que os mesmos precisem escrever complicadas consultas SQL ou mesmo entender o funcionamento do modelo de banco de dados para tal.

7. REFERÊNCIA BIBLIOGRÁFICA

Camanho, M.M. **Conciliando Coerência E Responsividade Em Storytelling Interativo**, MSc thesis, Departamento de Informática, Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro (2009)

Camanho, M.M.; Ciarlini, A.E.M.; Furtado, A.L.; Pozzer, C.T.; Feijó, B. **A Model for Interactive TV Storytelling**. In Anais do VIII Simpósio Brasileiro de Jogos e Entretenimento Digital SBGAMES 200, Rio de Janeiro, outubro de 2009.

Ciarlini, A. E. M.. **Geração interativa de enredos**. PhD dissertation, Departamento de Informática, PUC-Rio, Rio de Janeiro (1999).

Ciarlini, A. E. M.; Pozzer, C. T.; Furtado, A. L. and Feijó, B. **A Logic-Based Tool for Interactive Generation and Dramatization of Stories**. In Proceedings of the ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE 2005), Valencia, pp.133-140, June 2005.

Dória, T. R. **Dramatização Não Determinística de Enredos em Storytelling para TV Interativa**, MSc thesis, Departamento de Informática, Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro (2009).

APÊNDICE 1. MAPEAMENTO DAS ENTIDADES

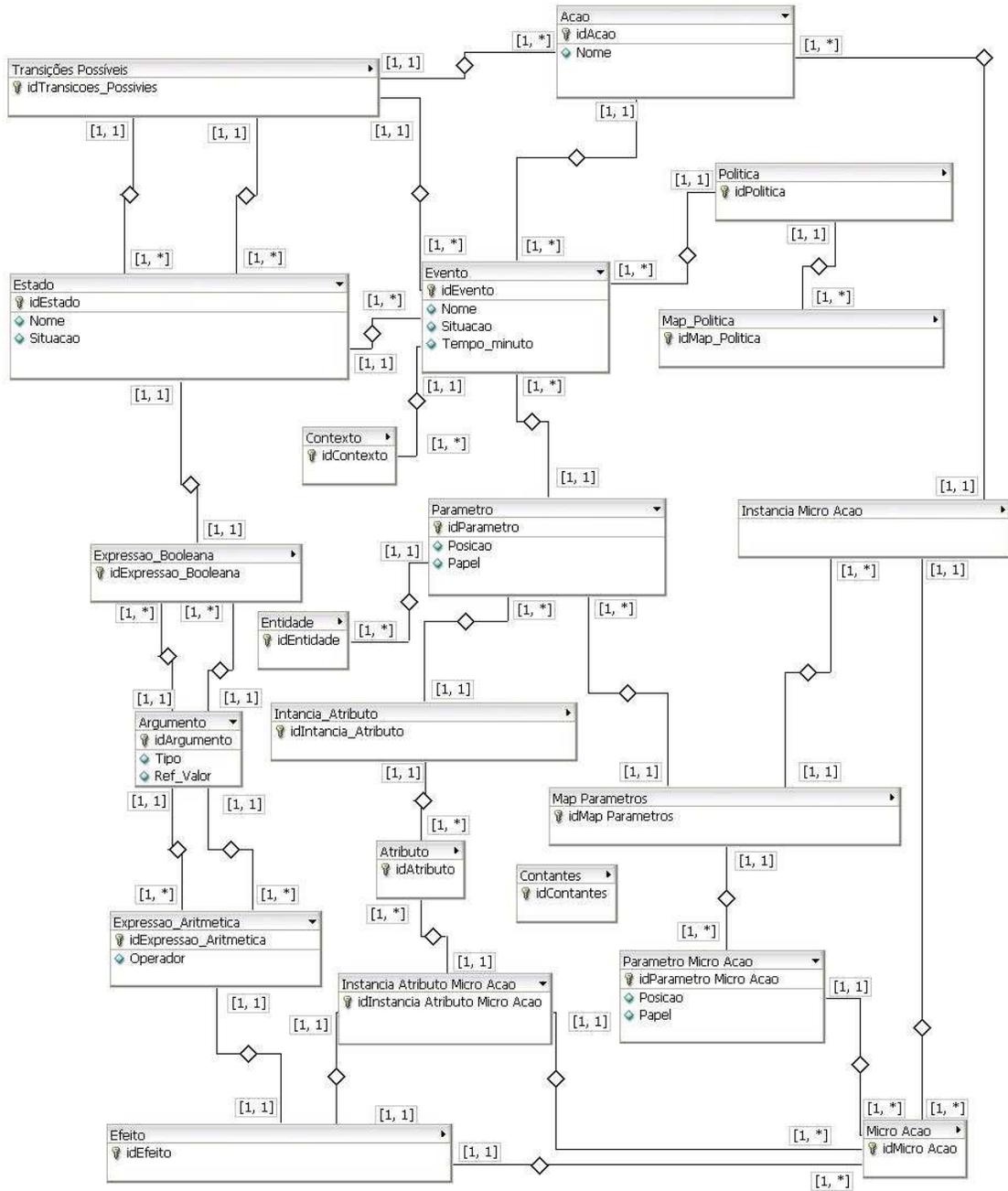


Figura 37: Modelo de dados Entidade-Relacionamento relativo à especificação de contextos.

As classes representadas no diagrama da Figura 37 mapeiam as tabelas do banco de dados da seguinte forma:

- **Login:** Classe que representa um login de usuário. Incluída para que futuramente contextos possam ser associados a usuários. Contém os seguintes atributos:
 - **Id:** Coluna “id”, que é o identificador da tabela “login”, cujo valor é gerado automaticamente a partir de um sequencial (editor_login_sequence).
 - **Username:** Coluna “login”, contém o nome de usuário.
 - **Password:** Coluna “password”, que contém a senha do usuário.

- **Context:** Classe que representa a tabela “contexto”, e contém os atributos:
 - **Id:** Coluna “id”, que é o identificador da tabela “contexto”, cujo valor é gerado automaticamente a partir de um sequencial (contexto_sequence).
 - **Name:** Coluna “nome”, que contém uma descrição textual do contexto.
 - **Events:** Conjunto de objetos da classe Event, representa a relação um-para-muitos existente com a tabela “evento”.

- **Event:** Classe que representa a tabela “evento”, e contém os seguintes atributos:
 - **Id:** Coluna “id”, que é o identificador da tabela “evento”, cujo valor é gerado automaticamente a partir de um sequencial (evento_sequence).
 - **Name:** Coluna “nome”, contendo uma descrição textual do evento.
 - **State:** Coluna “situacao”, contendo informação textual sobre a situação do evento.
 - **Duration:** Coluna “tempo_minutos”, contendo um valor numérico referente à duração do evento.

- **Context:** Coluna “id_contexto”, representado por um objeto da classe Context, que corresponde à relação muitos-para-um com a tabela “contexto”.
 - **Parameters:** Conjunto de objetos da classe Parameter, representa a relação um-para-muitos existente com a tabela “parametro”.
 - **States:** Conjunto de objetos da classe State, representa a relação um-para-muitos existente com a tabela “estado”.
 - **Actions:** Conjunto de objetos da classe Action, representa a relação um-para-muitos existente com a tabela “acao”.
- **State:** Classe que representa a tabela “estado”, e contém os seguintes atributos:
 - **Id:** Coluna “id”, que é o identificador da tabela “estado”, cujo valor é gerado automaticamente a partir de um sequencial (estado_sequence).
 - **Name:** Coluna “nome”, contendo uma descrição textual do estado.
 - **Situation:** Coluna “situacao”, contendo uma informação textual sobre a situação do estado.
 - **Event:** Coluna “id_evento”, representado por um objeto da classe Event, que corresponde à relação muitos-para-um com a tabela “evento”.
 - **BooleanExpression:** Coluna “id_expressao_booleana”, representado por um objeto da classe BooleanExpression, que corresponde à relação muitos-para-um com a tabela “expressao_booleana”. Trata-se da expressão booleana aplicada sobre os valores dos parâmetros, e utilizada para verificar se o evento encontra-se no estado especificado.
- **Parameter:** Classe que representa a tabela “parametro”, e contém os seguintes atributos:

- **Id:** Coluna “id”, que é o identificador da tabela “parametro”, cujo valor é gerado automaticamente a partir de um sequencial (parametro_sequence).
 - **Position:** Coluna “posicao”, contendo um valor numérico representando a ordenação do parâmetro no evento.
 - **Role:** Coluna “papel”, contendo descrição textual do papel desempenhado por esse parâmetro no evento.
 - **Event:** Coluna “id_evento”, representado por um objeto da classe Event, que corresponde à relação muitos-para-um com a tabela “evento”.
 - **EntityType:** Coluna “id_tipo_entidade”, representado por um objeto da classe EntityType, que corresponde à relação muitos-para-um com a tabela “entidade”.
 - **AttributeInstances:** Conjunto de objetos da classe AttributeInstance, representa a relação muitos-para-muitos existente com a tabela “atributo” através da tabela “instancia_atributo”. Cada parâmetro pode ter múltiplos atributos, e essas instâncias de atributo podem ser utilizadas como argumentos de expressões booleanas e aritméticas no Logtell.
- **EntityType:** Classe que representa a tabela “entidade”, e contém os seguintes atributos:
 - **Id:** Coluna “id”, que é o identificador da tabela “entidade”, cujo valor é gerado automaticamente a partir de um sequencial (entidade_sequence).
 - **Name:** Coluna “nome”, contendo uma descrição textual do tipo de entidade.
 - **Attribute:** Classe que representa a tabela “atributo”, e contém os seguintes atributos:

- **Id:** Coluna “id”, que é o identificador da tabela “atributo”, cujo valor é gerado automaticamente a partir de um sequencial (atributo_sequence).
 - **Name:** Coluna “nome”, contendo uma descrição textual do atributo.
 - **Type:** Coluna “tipo”, contendo uma informação textual do tipo de valor do atributo.
 - **EntityType:** Coluna “id_entidade”, representado por um objeto da classe EntityType, que corresponde à relação muitos-para-um com a tabela “entidade”. Trata-se do tipo de entidade do parâmetro ao qual esse atributo se aplica.
- **Action:** Classe que representa a tabela “acao”, e contém os seguintes atributos:
 - **Id:** Coluna “id”, que é o identificador da tabela “acao”, cujo valor é gerado automaticamente a partir de um sequencial (acao_sequence).
 - **Name:** Coluna “nome”, contendo uma descrição textual da ação.
 - **Duration:** Coluna “duracao_segundos”, contendo um valor numérico indicando a duração da ação.
 - **Event:** Coluna “id_evento”, representado por um objeto da classe Event, que corresponde à relação muitos-para-um com a tabela “evento”.
 - **Instances:** Conjunto de objetos da classe MicroActionInstance, representa a relação um-para-muitos existente com a tabela “instancia_micro_acao”. Cada ação pode ser executada através de uma sequencia de várias micro-ações, que por sua vez possui instâncias para cada ação que dependa delas.
- **MicroAction:** Classe que representa a tabela “micro_acao”, e contém os seguintes atributos:

- **Id:** Coluna “id”, que é o identificador da tabela “micro_acao”, cujo valor é gerado automaticamente a partir de um sequencial (micro_acao_sequence).
 - **Name:** Coluna “nome”, contendo uma descrição textual da micro-ação.
 - **Duration:** Coluna “duracao_segundos”, contendo um valor numérico indicando a duração da micro-ação.
 - **MicroActionParameters:** Conjunto de objetos da classe MicroActionParameter, representa a relação um-para-muitos existente com a tabela “parametro_micro_acao”. Trata-se dos parâmetros necessários para que a micro-ação seja executada.
 - **MicroActionPreConditions:** Conjunto de objetos da classe MicroActionParameter, representa a relação um-para-muitos existente com a tabela “pre_condicao_micro_acao”.
- **MicroActionPreCondition:** Classe que representa a tabela “pre_condicao_micro_acao”, e contém os seguintes atributos:
 - **Id:** Coluna “id”, que é o identificador da tabela “pre_condicao_micro_acao”, cujo valor é gerado automaticamente a partir de um sequencial (pre_condicao_micro_acao_sequence).
 - **MicroAction:** Coluna “id_micro_acao”, representado por um objeto da classe MicroAction, que corresponde à relação muitos-para-um com a tabela “micro_acao”.
 - **BooleanExpression:** Coluna “id_expressao_booleana”, representado por um objeto da classe BooleanExpression, que corresponde à relação muitos-para-um com a tabela “expressao_booleana”. Trata-se da expressão booleana a ser testada para a micro-ação ao qual a pré-condição está associada.

- **MicroActionParameter:** Classe que representa a tabela “parametro_micro_acao”, e contém os seguintes atributos:
 - **Id:** Coluna “id”, que é o identificador da tabela “parametro_micro_acao”, cujo valor é gerado automaticamente a partir de um sequencial (parametro_micro_acao_sequence).
 - **Role:** Coluna “papel”, contendo uma informação textual sobre o papel desempenhado pelo parâmetro de micro-ação.
 - **Position:** Coluna “posicao”, contendo um valor numérico representando a ordenação do parâmetro na micro-ação.
 - **MicroAction:** Coluna “id_micro_acao”, representado por um objeto da classe MicroAction, que corresponde à relação muitos-para-um com a tabela “micro_acao”.
 - **EntityType:** Coluna “id_entidade”, representado por um objeto da classe EntityType, que corresponde à relação muitos-para-um com a tabela “entidade”.

- **MicroActionInstance:** Classe que representa a tabela “instancia_micro_acao”, e contém os seguintes atributos:
 - **Id:** Coluna “id”, que é o identificador da tabela “instancia_micro_acao”, cujo valor é gerado automaticamente a partir de um sequencial (instancia_micro_acao_sequence).
 - **Name:** Coluna “nome”, contendo uma descrição textual da instância de micro-ação.
 - **Action:** Coluna “id_acao”, representado por um objeto da classe Action, que corresponde à relação muitos-para-um com a tabela “acao”.
 - **MicroAction:** Coluna “id_micro_acao”, representado por um objeto da classe MicroAction, que corresponde à relação muitos-para-um com a tabela “micro_acao”.

- **Parent:** Coluna “id_instancia_micro_acao_pai”, representado por um objeto da classe `MicroAction`, que corresponde à relação muitos-para-um com a tabela “micro_acao”. Esta coluna define uma hierarquia de instâncias de micro-ação, definindo uma ordem de execução para as mesmas.
 - **ParameterMapping:** Conjunto de objetos da classe `ParameterMap`, representa a relação um-para-muitos existente com a tabela “map_parametros”. Mapeia os parâmetros do evento ao qual a ação pertence aos parâmetros necessários para a execução na micro-ação.
 - **Effects:** Conjunto de objetos da classe `Effect`, representa a relação um-para-muitos existente com a tabela “efeito”. Trata-se dos efeitos causados pela execução da micro-ação sobre o estado dos parâmetros do evento.
- **Effect:** Classe que representa a tabela “efeito”, e contém os seguintes atributos:
 - **Id:** Coluna “id”, que é o identificador da tabela “efeito”, cujo valor é gerado automaticamente a partir de um sequencial (efeito_sequence).
 - **MicroAction:** Coluna “id_micro_acao”, representado por um objeto da classe `MicroAction`, que corresponde à relação muitos-para-um com a tabela “micro_acao”.
 - **MicroActionInstance:** Coluna “id_instancia_micro_acao”, representado por um objeto da classe `MicroActionInstance`, que corresponde à relação muitos-para-um com a tabela “instancia_micro_acao”.
 - **ArithmeticExpression:** Coluna “id_expressao_aritmetica”, representado por um objeto da classe `ArithmeticExpression`, que corresponde à relação muitos-para-um com a tabela “expressao_aritmetica”. Trata-se da expressão aritmética a ser executada sobre os parâmetros da micro-ação associada.

- **ParameterMap:** Classe que representa a tabela “map_parametros”, e contém os seguintes atributos:
 - **Id:** Coluna “id”, que é o identificador da tabela “map_parametros”, cujo valor é gerado automaticamente a partir de um sequencial (map_parametros_sequence).
 - **MicroActionInstance:** Coluna “id_instancia_micro_acao”, representado por um objeto da classe MicroActionInstance, que corresponde à relação muitos-para-um com a tabela “instancia_micro_acao”.
 - **Parameter:** Coluna “id_parametro_evento”, representado por um objeto da classe Parameter, que corresponde à relação muitos-para-um com a tabela “parametro”.
 - **MicroActionParameter:** Coluna “id_parametro_micro_acao”, representado por um objeto da classe MicroActionParameter, que corresponde à relação muitos-para-um com a tabela parametro_micro_acao”.

- **Argument:** Classe que representa a tabela “argumento”. Esta tabela possui duas colunas, “tipo” e “ref_valor”, que indicam a tabela e o identificador da linha a qual este argumento está associado. A classe Argument possui os seguintes atributos:
 - **Id:** Coluna “id”, que é o identificador da tabela “argumento”, cujo valor é gerado automaticamente a partir de um sequencial (argumento_sequence).
 - **Type:** Coluna “tipo”, contendo uma descrição textual sobre o tipo do argumento. Os tipos válidos estão listados a seguir:

| Tipo | Classe |
|-------------------------------|----------------------|
| constante | Constant |
| constante_string | StringConstant |
| expressao_booleana | BooleanExpression |
| instancia_atributo | AttributeInstance |
| instancia_atributo_micro_acao | ArithmeticExpression |

- **ReferenceValue:** Coluna “ref_valor”, contendo um valor numérico.
 - **AbstractArgument:** Este atributo não mapeia uma coluna da tabela “argumento”, e é utilizado para representar o objeto referenciado pela combinação de “tipo” e “ref_valor”.
- **AbstractArgument:** Esta classe não está associada a nenhuma tabela do banco de dados. Trata-se de uma superclasse estendida por subclasses que tem em comum a possibilidade de serem utilizadas como argumentos em expressões booleanas ou aritméticas. Estas expressões, inclusive, também estendem AbstractArgument.
 - **Constant:** Esta classe não está associada a nenhuma tabela do banco de dados. Quando a linha da tabela “argumento” possui a coluna “tipo” contendo o valor “constante”, a coluna “ref_valor” possui o valor numérico da constante. Essa classe possui o seguinte atributo:
 - **Value:** Informação numérica contendo o valor da constante.
 - **StringConstant:** Subclasse de AbstractArgument que representa a tabela “constante”. Quando a linha da tabela “argumento” possui a coluna “tipo”

contendo o valor “constante_string”, a coluna “ref_valor” possui o identificador da tabela “constante”. Esta classe possui os seguintes atributos:

- **Id:** Coluna “id”, que é o identificador da tabela “constante”, cujo valor é gerado automaticamente a partir de um sequencial (constante_sequence).
- **Value:** Informação textual contendo o valor da constante.

- **BooleanExpression:** Subclasse de AbstractArgument que representa a tabela “expressao_booleana”. Quando a linha da tabela “argumento” possui a coluna “tipo” contendo o valor “expressao_booleana”, a coluna “ref_valor” possui o identificador da tabela “expressao_booleana”. Esta classe possui os seguintes atributos:

- **Id:** Coluna “id”, que é o identificador da tabela “expressao_booleana”, cujo valor é gerado automaticamente a partir de um sequencial (expressao_booleana_sequence).
- **Operator:** Coluna “operador”, contendo uma informação textual representando o operador da expressão booleana.
- **Argument1:** Coluna “id_argumento1”, representado por um objeto da classe Argument, que corresponde à relação muitos-para-um com a tabela “argumento”. Trata-se do primeiro argumento da expressão booleana.
- **Argument2:** Coluna “id_argumento2”, representado por um objeto da classe Argument, que corresponde à relação muitos-para-um com a tabela “argumento”. Trata-se do segundo argumento da expressão booleana.
- **AbstractArgument1:** Este atributo não mapeia uma coluna da tabela “expressao_booleana”, e é utilizado somente para referenciar o objeto AbstractArgument contido em Argument1.

- **AbstractArgument2:** Este atributo não mapeia uma coluna da tabela “expressao_booleana”, e é utilizado somente para referenciar o objeto AbstractArgument contido em Argument2.

- **ArithmeticExpression:** Subclasse de AbstractArgument que representa a tabela “expressao_aritmetica”. Quando a linha da tabela “argumento” possui a coluna “tipo” contendo o valor “instancia_atributo_micro_acao”, a coluna “ref_valor” possui o identificador da tabela “expressao_aritmetica”. Esta classe possui os seguintes atributos:
 - **Id:** Coluna “id”, que é o identificador da tabela “expressao_aritmetica”, cujo valor é gerado automaticamente a partir de um sequencial (expressao_aritmetica_sequence).
 - **Operator:** Coluna “operador”, contendo uma informação textual representando o operador da expressão aritmética.
 - **Argument1:** Coluna “id_argumento1”, representado por um objeto da classe Argument, que corresponde à relação muitos-para-um com a tabela “argumento”. Trata-se do primeiro argumento da expressão aritmética.
 - **Argument2:** Coluna “id_argumento2”, representado por um objeto da classe Argument, que corresponde à relação muitos-para-um com a tabela “argumento”. Trata-se do segundo argumento da expressão aritmética.
 - **AbstractArgument1:** Este atributo não mapeia uma coluna da tabela “expressao_aritmetica”, e é utilizado somente para referenciar o objeto AbstractArgument contido em Argument1.
 - **AbstractArgument2:** Este atributo não mapeia uma coluna da tabela “expressao_aritmetica”, e é utilizado somente para referenciar o objeto AbstractArgument contido em Argument2.

- **AttributeInstance:** Subclasse de AbstractArgument que representa a tabela “instancia_atributo”. Quando a linha da tabela “argumento” possui a coluna “tipo” contendo o valor “instancia_atributo”, a coluna “ref_valor” possui o identificador da tabela “instancia_atributo”. Esta classe possui os seguintes atributos:
 - **Id:** Coluna “id”, que é o identificador da tabela “instancia_atributo”, cujo valor é gerado automaticamente a partir de um sequencial (instancia_atributo_sequence).
 - **Parameter:** Coluna “id_parametro”, representado por um objeto da classe Parameter, que corresponde à relação muitos-para-um com a tabela “parametro”.
 - **Attribute:** Coluna “id_atributo”, representado por um objeto da classe Attribute, que corresponde à relação muitos-para-um com a tabela “atributo”.