Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

Sistema Especialista de Inteligência Artificial Integrado a Plataforma WEB para Entretenimento em um Ambiente Interativo Multiusuário

Autor:

Daniel Chaves Toscano Dantas

Autor:

Eliseu Paz e Silva de Guimarães

Orientador:

Prof. Sérgio Barbosa Villas Boas, Ph. D.

Examinador:

Prof. Aloysio de Castro Pinto Pedroza

Examinador:

Prof. Antônio Cláudio Gómez de Sousa

DEL

Agosto de 2010

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica – Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro – RJ CEP 21949-900

Este exemplar é de propriedade da Universidade Federal do Rio de Janeiro, que poderá incluí-lo em base de dados, armazenar em computador, microfilmar ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do(s) autor(es) e do(s) orientador(es).

DEDICATÓRIA

Dedico este trabalho aos meus pais que sempre me apoiaram em todas as minhas necessidades. Dedico este trabalho aos professores que contribuíram para minha formação como engenheiro e como ser humano apto para exercer sua profissão. Dedico este trabalho aos amigos que fiz na faculdade e que fizeram com que este período nunca seja apagado de minha memória. Dedico àqueles que apesar de todos os problemas e dificuldades sempre souberam valorizar a universidade pública.

Daniel Chaves Toscano Dantas

Dedico este trabalho à minha mãe, que já há algum tempo tem sido o meu Norte. Dedico também a meu avô, Salomão, que infelizmente não pôde ver o neto se formar “na Federal” e que me ensinou o verdadeiro significado da palavra perseverança.

Eliseu Paz e Silva de Guimarães

AGRADECIMENTO

Agradeço aos meus pais que sempre me apoiaram de todas as formas possíveis. Agradeço aos meus amigos e familiares. Agradeço aos professores que fizeram com que este trabalho se tornasse possível. Agradeço ao meu amigo Eliseu por todos os bons momentos que passamos durante todo este período.

Este trabalho é para todos vocês.

Daniel Chaves Toscano Dantas

Agradeço aos professores e funcionários que fazem da Universidade do Brasil uma instituição de ensino superior público e de qualidade. Agradeço a todos aqueles que, com o suor de seu trabalho, e mesmo que nunca saibam, permitem a tantos este ensino.

Agradeço aos meus amigos, à minha ex-namorada e à minha família, que entenderam ou pelo menos aceitaram os momentos de ausência. Momentos que permitiram a conclusão de meu curso e deste trabalho.
Agradeço ao Daniel, a melhor dupla que eu poderia querer.

Eliseu Paz e Silva de Guimarães

RESUMO

O trabalho corresponde a um jogo com interface WEB. Neste jogo o usuário é responsável pela administração de um time de futebol. Este time de futebol disputa campeonatos com times criados por outros usuários. Um diferencial deste jogo é que os usuários podem visualizar as partidas entre seus times. Para isso foi desenvolvido um sistema especialista de inteligência artificial que cria agentes inteligentes capazes de simular jogadores reais durante uma partida de futebol.

Palavras-chave: sistema web, inteligência artificial, entretenimento.

ABSTRACT

This paper describes a Web Interface game. In this game the user is responsible to manage a soccer team. This team plays in leagues with teams created by another users. One big difference is that in this game the users are allowed to watch the matches already played. To make this possible was developed an artificial inteligence expert algorithm that controls autonomous agents, making them look like real players during the match.

Key-words: WEB system, artificial intelligence, entertainment.

SIGLAS

UFRJ – Universidade Federal do Rio de Janeiro

MEF – Máquina de Estado Finita

IA – Inteligência Artificial

BD – Banco de Dados

DER – Diagrama de Entidades e Relacionamentos

SGBD – Sistema de Gerenciamento de Banco de Dados

DA – *Data Access*

DAO – *Data Access Object*

WS – *Web Services*

WSDL – *Web Services Description Language*

XML – *Extensible Markup Language*

JSF – *Java Server Faces*

MVC – *Model-View-Controller*

HTML – *Hyper Text Markup Language*

CSS – *Cascading Style Sheets*

XHTML – *Extensible Hypertext Markup Language*

JAX-WS – *Java API for XML Web Services*

JSON – *JavaScript Object Notation*

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 – Tema

O tema do trabalho é entretenimento na WEB utilizando um sistema especialista de Inteligência Artificial. Trata-se de uma plataforma de entretenimento na WEB onde cada usuário assumirá o controle de um time de futebol virtual. Este usuário será responsável por toda a parte gerencial incluindo-se a gestão financeira, manutenção patrimonial, política de contratações, treinamento e escalação do time. O usuário poderá assistir às partidas do seu time em "tempo real" vendo o desempenho de cada jogador. A visualização de cada jogo será feita por um sistema especialista de inteligência artificial que será responsável por toda e qualquer interação, movimento ou tomada de decisão ocorrida durante a partida de futebol.

1.2 – Delimitação

Atualmente os sistemas WEB encontram-se plenamente difundidos e cada vez mais presentes na vida das pessoas. Sendo o entretenimento uma das maiores fontes de receita do mundo da computação os jogos online se tornaram em pouco tempo uma grande febre na internet devido ao seu poder de divertir e integrar através de largas distâncias. Sistemas de entretenimento que além de divertir e integrar sejam ainda capazes de tomar decisões e simular complexos padrões comportamentais são atualmente alvo de intensa pesquisa tanto pelo meio acadêmico como pela indústria de alta tecnologia.

1.3 – Justificativa

Existem quatro módulos disjuntos no trabalho que serão justificados separadamente.

Módulo de Banco de Dados:

A persistência de dados é pré-requisito básico para qualquer sistema desenvolvido atualmente. Não há maneira de se pensar em algum tipo de atividade que gere informações e que prescinda de guardá-las de alguma forma. O armazenamento, a organização e a recuperação de dados têm sido, então, uma área bastante explorada em sistemas de computação de todo o tipo. A garantia de integridade dos dados e uma fácil acessibilidade têm se mostrados desafios cada vez mais fáceis de serem superados, com o desenvolvimento de técnicas e ferramentas de gerenciamento de bancos de dados.

Módulo WEB:

A existência de um módulo WEB é de fundamental importância no sistema, uma vez que é a partir dele que o usuário final poderá ver o resultado de todo o processamento feito pelos demais módulos do sistema. Em outras palavras, podemos dizer que toda a camada WEB tem por finalidade a apresentação para o usuário, objetivo primordial de um sistema que tem por meta o entretenimento.

Módulo de Inteligência artificial:

Esta área é uma das mais desafiantes existentes desde o tempo dos primeiros computadores. O desejo humano de reproduzir seu comportamento em uma máquina ocupou a mente de inúmeros cientistas, engenheiros e filósofos desde os mais remotos tempos. Nunca se obteve tantos avanços no campo da inteligência artificial como nas últimas duas décadas. O futuro que construímos hoje contará com máquinas que substituirão os homens nas tarefas em que se exige frieza, precisão e rapidez. Haverá máquinas tomando decisões todo o tempo. Desde simples guias turísticos até cirurgiões e nano dispositivos capazes de combater bactérias durante um ataque biológico.

Módulo de Web Services:

A necessidade de existência de um módulo de WS se dá ao passo que o sitema prevê integração de diferentes módulos. O uso de WS é, por vezes, fundamental na integração entre estas partes disjuntas, favorecendo a universalidade e a padronização das informações trocadas entre os diferentes módulos.

1.4 – Objetivos

O Objetivo do trabalho é produzir um ambiente WEB multiusuário onde todos possam participar de um campeonato de futebol com times virtuais. Cada usuário terá total controle sobre a administração do seu time e poderá assistir aos jogos contra equipes de outros usuários. O objetivo a ser atingido é mostrar a integração de áreas de conhecimento totalmente disjuntas como BD, Programação para WEB, WS e IA através de um sistema WEB totalmente integrado e projetado para promover o entretenimento. Serão considerados como objetivos importantes no trabalho os seguintes módulos:

Banco de Dados: Aqui será demonstrada toda a parte de banco de dados, que tem como funções principais:

1) Geração de um banco de dados com as características necessárias ao funcionamento do sistema.
2) Armazenamento dos dados gerados por comandos do usuário ou pelo sistema de inteligência artificial.
3) Recuperação de dados, requisitados pelo usuário ou pelo sistema de inteligência artificial.

Interface WEB: Aqui será demonstrada a interface WEB à disposição do usuário. Esta interface consiste em duas categorias independentes:

1) Menus administrativos do usuário que contém todas as opções disponíveis para o gerenciamento de seu time
2) Interface Applet que é o meio através do qual o usuário visualiza o jogo em andamento.

Sistema de Inteligência Artificial: Aqui será demonstrado de forma simples como produzir a sensação de autonomia e inteligência em agentes computacionais. Estes agentes produzirão a sensação de trabalho em equipe e terão como objetivo vencer a partida contra o time adversário.

Web Services: Aqui será demonstrada a lógica de desenvolvimento dos WS e a sua importância na integração de diferentes aplicativos. Serão também abordados os conceitos e as tecnologias utilizados para o bom desenvolvimento deste módulo.

1.5 – Metodologia

Para se alcançar o desenvolvimento de forma satisfatória de um sistema especialista da complexidade do aqui pretendido, é necessário o entendimento prévio de todas as suas partes disjuntas e, posteriormente, a adequação delas para que formem um sistema final coeso e eficiente.

O entendimento de cada uma das partes dá-se, inicialmente, a partir de conhecimentos pré-adquiridos ao longo do curso de Engenharia Eletrônica e de computação desta universidade. Contudo, ao passo que o desenvolvimento de um sistema de tal tipo demanda certas particularidades, são necessárias pesquisas de cada parte que o compõem. Estas pesquisas têm como objetivo fundamental fornecer subsídios para que o desenvolvimento do sistema se dê de forma consistente, assegurando sua confiabilidade.

Mais do que apenas garantir o seu funcionamento, espera-se que ele, por se voltar para o entretenimento, tenha características de fácil acesso ao usuário final, por vezes não acostumado a demandas de conhecimento técnico. As pesquisas se mostram, também neste aspecto, importantes: o produto gerado deve ser facilmente manipulável por não acadêmicos, o que aproxima o seu desenvolvimento de áreas muitas vezes distantes da parte tecnológica. É necessário que se imagine a sensação de cada usuário do jogo, as maneiras de melhorá-lo, tanto no sentido da diversão quanto da acessibilidade.

Uma vez estabelecidas as necessidades técnicas e funcionais do sistema, todo o estudo de como desenvolvê-lo se volta neste sentido. É esta lista de prioridades que deve nortear o projeto, fazendo com que seu desenvolvimento se dê de forma a assegurar o cumprimento de prazos e, sobretudo, o seu perfeito funcionamento.

O aprofundamento no conhecimento das diversas áreas que integram o projeto passa a ser, então, direcionado para que se atenda às suas particularidades. Não deve haver gasto desnecessário de tempo em estudos que fujam do enfoque principal, mesmo

que muitas vezes isto signifique não buscar a totalidade de aprofundamento que seria possível.

A cada novo conhecimento adquirido são realizados testes para que se assegure a eficiência e o real entendimento do novo artefato a ser usado. Estes testes têm o objetivo de garantir a integridade do sistema após a introdução do artefato.

Portanto, esta introdução gradual dos artefatos possibilita o desenvolvimento do sistema em várias iterações. Cada uma delas é composta das seguintes etapas:

1) Brainstorm.
2) Estimação de prazos e dificuldades.
3) Design.
4) Codificação.
5) Testes.

Estas iterações contavam, cada uma, com uma lista de itens a serem implementados. As iterações realizadas para o desenvolvimento do sistema foram, basicamente:

1) Modelagem de dados.
2) Especificação de requisitos – Implantação do BD.
3) Desenvolvimento das classes de controle e acesso ao BD – Desenvolvimento do módulo IA.
4) Desenvolvimento do WS – Desenvolvimento do applet.
5) Desenvolvimento da interface WEB.
6) Integração dos módulos.

Nos capítulos que se seguem serão detalhados alguns dos conhecimentos adquiridos para o desenvolvimento deste sistema, bem como suas aplicações específicas para o caso aqui estudado.

1.6 – Descrição

O capítulo 2 mostrará toda a parte referente ao Banco de Dados. Ele conterá a especificação detalhada do Diagrama de Entidades e Relacionamentos que compõe o banco. Além disso, haverá neste capítulo a explicação da forma de criação do BD, sua manutenção e a forma de acesso a ele. Ainda neste capítulo haverá o detalhamento de

cada método que faz alterações no BD, sua necessidade, funcionalidade e funcionamento.

O capitulo 3 conterá toda a especificação referente à interface WEB e nele será explicado em detalhes toda e qualquer tecnologia utilizada, bem como a razão de cada item exposto na interface. Haverá uma discussão sobre tecnologias existentes e quais motivos levaram às escolhas que foram feitas.

O capítulo 4 explicará nos mínimos detalhes as técnicas utilizadas para produzir a ilusão de inteligência durante a partida de futebol. Haverá uma discussão sobre a física utilizada, sobre a política de tomada de decisão e sobre toda a complexa maquina de estados que rege toda a interação entre os agentes inteligentes durante o curso da partida.

O capítulo 5 traz explicações referentes ao desenvolvimento dos WS. O ponto crucial deste capítulo é a demonstração de como diferentes tecnologias, protocolos e padrões servem a um único propósito: a integração de sistemas.

Ao final deste documento, será possível entender-se a ligação entre os diferentes módulos, conforme a figura abaixo.

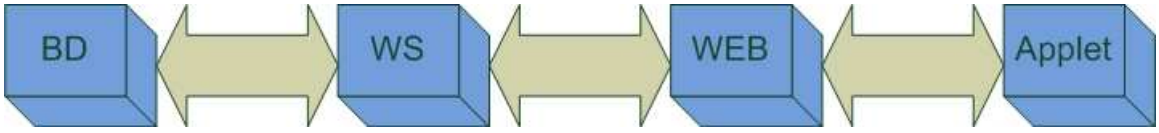

Figura 1.1 – Relacionamento entre os módulos do sistema.

Capítulo 2

Banco de Dados

2.1 – Descrição

Neste capítulo será apresentado o BD referente ao sistema, desde a sua concepção até os métodos que permitem o acesso dos outros módulos aos dados armazenados nele.

Serão explicitadas as entidades do modelo do sistema, para que se entenda a função de cada uma delas, além de serem especificados todos os seus atributos. Os relacionamentos entre as entidades também serão apresentados, bem como sua importância para a implantação de toda a regra de negócio do sistema.

Além disso, os métodos que permitem o acesso ao banco também serão explicitados, com a descrição de seus parâmetros de entrada.

2.2 – DER

A figura abaixo mostra o DER do sistema.

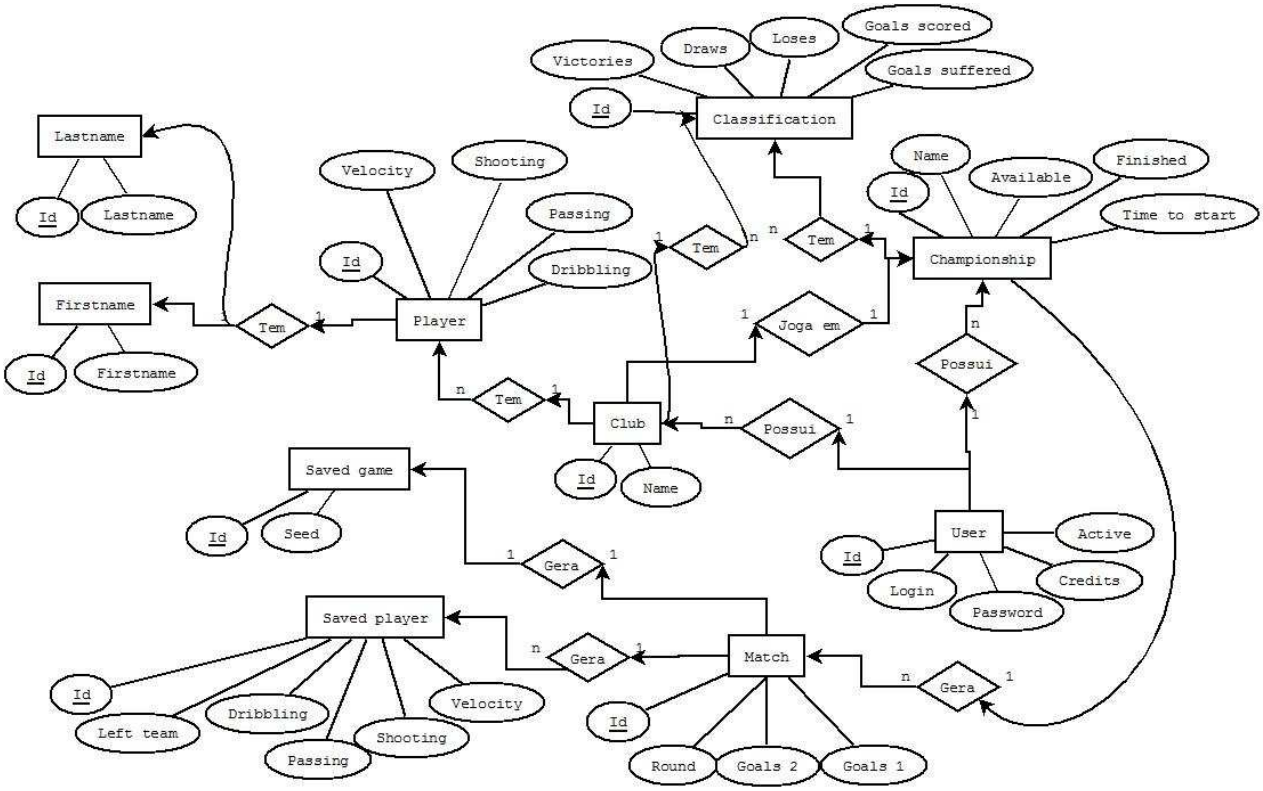

Figura 2.1 – DER

Abaixo estão explicitadas as entidades do diagrama e seus atributos. Cada uma destas entidades dará origem a uma tabela no BD.

2.2.1 – Championships

Esta entidade tem como atributos as informações básicas referentes aos campeonatos criados através do sistema desenvolvido. A tabela abaixo especifica estes atributos.

Atributo	Tipo
Championship_id	Serial
Championship_creator_id	Serial
Championship_name	Character Varying
Available	Boolean
Finished	Boolean
Time_to_start	Integer

Tabela 2.1 – Atributos da entidade Championships

2.2.2 – Classifications

Esta entidade tem como atributos as informações sobre o desempenho de um determinado time em um determinado campeonato. A tabela abaixo especifica estes atributos.

Atributo	Tipo
Classification_id	Serial
Championship_id	Serial
Club_id	Serial
Victories	Integer
Draws	Integer
Loses	Integer
Goals_scored	Integer
Goals_suffered	Integer

Tabela 2.2 – Atributos da entidade Classifications

2.2.3 – Clubs

Esta entidade tem como atributos as informações básicas sobre um time. A tabela abaixo especifica estes atributos.

Atributo	Tipo
Club_id	Serial
Club_owner_id	Serial
Club_name	Character Varying

Tabela 2.3 – Atributos da entidade Clubs

2.2.4 – Firstnames

Esta entidade tem como atributos os nomes que podem ser atribuídos aos jogadores dos times. A tabela abaixo especifica estes atributos.

Atributo	Tipo
Firstname_id	Serial
Firstname	Character Varying

Tabela 2.4 – Atributos da entidade Firstnames

2.2.5 – Lastnames

Esta entidade tem como atributos os sobrenomes que podem ser atribuídos aos jogadores dos times. A tabela abaixo especifica estes atributos.

Atributo	Tipo
Lastname_id	Serial
Lastname	Character Varying

Tabela 2.5 – Atributos da entidade Lastnames

2.2.6 – Matches

Esta entidade tem como atributos as informações básicas referentes a uma partida disputada por dois times através deste sistema. A tabela abaixo especifica estes atributos.

Atributo	Tipo
Match_id	Serial
Championship_id	Serial
Club_1_id	Serial
Club_2_id	Serial
Goals_1	Integer
Goals_2	Integer
Round	Integer

Tabela 2.6 – Atributos da entidade Matches

2.2.7 – Players

Esta entidade tem como atributos as informações sobre os jogadores, bem como suas características. A tabela abaixo especifica estes atributos.

Atributo	Tipo
Player_id	Serial
Club_id	Serial
Firstname	Character Varying
Lastname	Character Varying
Dribbling	Integer
Passing	Integer
Shooting	Integer
Velocity	Integer

Tabela 2.7 – Atributos da entidade Players

2.2.8 – Saved_games

Esta entidade tem como atributos parte das informações necessárias para se recuperar uma partida. A tabela abaixo especifica estes atributos.

Atributo	Tipo
Saved_game_id	Serial
Seed	Bigint

Tabela 2.8 – Atributos da entidade Saved_games

2.2.9 – Saved_players

Esta entidade tem como atributos a parte das informações dos jogadores, que são necessárias para se recuperar uma partida. A tabela abaixo especifica estes atributos.

Atributo	Tipo
Saved_player_id	Serial
Saved_game_id	Serial
Player_id	Serial
Left_team	Boolean
Dribbling	Integer
Passing	Integer
Shooting	Integer
Velocity	Integer

Tabela 2.9 – Atributos da entidade Saved_players

2.2.10 – Users

Esta entidade tem como atributos as informações sobre os usuários cadastrados no sistema. A tabela abaixo especifica estes atributos.

Atributo	Tipo
User_id	Serial
Login	Character Varying
Password	Character Varying
Credits	Integer
Active	Boolean

Tabela 2.10 – Atributos da entidade Users

2.3 – Chaves do BD

Há dois tipos de chaves presentes no BD: as primárias e as estrangeiras.

2.3.1 – Chaves primárias

Cada tabela do BD tem exatamente uma chave primária, que tem como única função estabelecer a unicidade de uma entrada no banco.

A tabela abaixo especifica as chaves do BD, associando cada uma à sua respectiva tabela.

Chave primária	Tabela
Championship_id	Championships
Classification_id	Classifications
Club_id	Clubs
Firstname_id	Firstnames
Lastname_id	Lastnames
Match_id	Matches
Player_id	Players
Saved_game_id	Saved_games
Saved_player_id	Saved_players
User_id	Users

Tabela 2.11 – Chaves primárias do BD

2.3.2 – Chaves estrangeiras

As chaves estrangeiras do BD têm o objetivo de associar as tabelas. Desta forma, estabelecemos de maneira adequada os relacionamentos entre as diferentes entidades do BD.

A tabela abaixo especifica as chaves estrangeiras, a tabela a que cada uma delas pertence e a tabela a que a chave se associa.

Chave estrangeira	Tabela de origem	Tabela associada
Championship_creator_id	Championships	Users
Championship_id	Classifications	Championships
Club_id		Clubs
Club_owner_id	Clubs	Users
Championship_id	Matches	Championships
Club_1_id		Clubs
Club_2_id		Clubs
Club_id	Players	Clubs
Saved_game_id	Saved_players	Saved_game
Player_id		Players

Tabela 2.12 – Chaves estrangeiras do BD

2.4 – Implantação do BD

2.4.1 – Usando SGBD

O BD foi implantado com a utilização de um SGBD. No caso o sistema escolhido foi o PostgreSQL, por ser livre, de fácil utilização e vasta documentação. A implantação foi feita tomando-se por base o DER, os atributos vistos nas tabelas de 2.1 a 2.10 e as chaves vistas nas tabelas 2.11 e 2.12.

O uso de um SGBD é de vital importância para que se assegure a integridade dos dados, a correção automática de falhas em um possível script de criação do banco e o acesso facilitado aos dados.

Esta implantação foi de fundamental importância para a realização dos testes do sistema, uma vez que era imprescindível que ele pudesse acessar o banco e fazer busca/inserção de dados desde o primeiro momento. A opção pelo não desenvolvimento de imediato de um software próprio para a criação do BD deve-se, basicamente, ao intuito de facilitar o início do trabalho, excluindo a alternativa do software próprio, caso ela se torne demasiadamente complexa ou até mesmo inviável.

2.5 – Acesso ao BD

De acordo com as boas práticas do desenvolvimento de sistemas, o acesso ao BD não pode ser feito diretamente por nenhum outro módulo que não aquele que foi desenvolvido especialmente para isso. Desta forma, foi criado um pacote especial somente para este acesso, chamado de Data Access (DA). A esquematização deste acesso pode ser vista na figura abaixo.

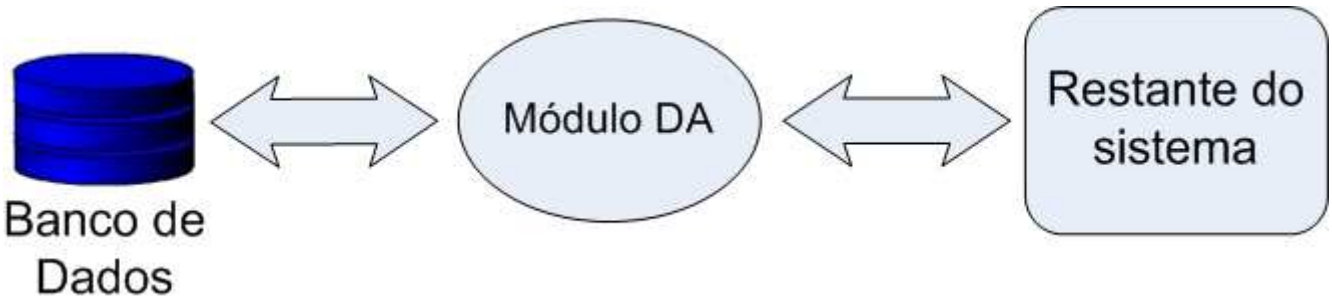

Figura 2.2 – Acesso ao banco usando DA.

2.5.1 – DA

O elemento mais básico do DA é aquele que se encarrega de fazer o acesso direto ao BD. Para isso, ele se utiliza de elementos de conexão já existentes em pacotes

específicos da linguagem JAVA. Através destes elementos são feitos o início e o fim da sessão, conectando-se e desconectando-se nossa aplicação do BD.

Para isso, neste elemento estão guardados o nome de usuário e a senha para acesso ao BD, além do nome do driver de acesso e do próprio nome do BD.

É também neste elemento que são definidas operações genéricas no banco, a saber:

1) Select – operação para selecionar um elemento do BD, através de um ou mais parâmetros pré-estabelecidos
2) Insert – operação para inserir uma entrada em uma determinada tabela do BD.
3) Update – operação para atualizar uma determinada entrada do BD.
4) Delete – operação para excluir uma determinada entrada do BD.

Todas estas operações possuem parâmetros que as tornam mais específicas, tais como a tabela na qual devem ser executadas, os valores existentes na entrada e possíveis elementos de comparação com dados existentes no BD.

Para cada tabela do BD é criada uma instância especial de DA, chamada de DAO. Abaixo são definidos os DAOs existentes neste sistema.

2.5.2 – DAOs

O acesso específico ao banco é feito através dos DAOs, que utilizam métodos existentes no DA genérico. Cada DAO possui as informações necessárias para o acesso a uma tabela do BD em particular.

Cada DAO possui métodos para recuperar entradas da tabela comparando valores vindos dos outros módulos do sistema com os valores existentes na tabela. Esta comparação pode se dar por igualdade ou por desigualdade, quando cabível tal nível de comparação. Tal operação é feita de maneira bastante simples, graças à existência do DA genérico. Com um pequeno trecho de código específico, presente no DAO, um mesmo método do DA pode servir para muitas tabelas e parâmetros distintos, reduzindo a duplicidade de código, eliminando redundâncias desnecessárias e assegurando que as falhas sejam mais facilmente identificáveis.

A inserção, a atualização e a exclusão de entradas no BD não são feitas de maneira tão simplificada, embora também haja métodos genéricos para estas operações

no DA. Isto porque, ao realizarmos estas operações, estamos alterando entradas do banco, o que pode gerar erros por tentativa de duplicidade de dados ou por exclusão de entradas relacionadas a entradas de outras tabelas. Todas estas possibilidades têm de ser analisadas, o que aumenta a complexidade da questão do acesso. Ainda assim, tomando-se como base as relações previamente estabelecidas pelo uso das chaves estrangeiras, esta análise torna-se tão eficiente quanto complexa, reduzindo a possibilidade de erro.

2.5.3 – Classes de tabelas

Para facilitar a manipulação dos dados, garantindo sua integridade tanto para observação quanto para alterações, foram criadas classes de tabelas.

As classes de tabelas são basicamente uma versão em código JAVA das tabelas existentes no BD. Podemos dizer que cada objeto de uma classe de tabela equivale a uma entrada do banco.

Existe, então, uma correlação entre os atributos das tabelas e os membros das classes. A idéia básica é que, ao selecionarmos uma entrada do BD, ela seja transformada em um objeto da classe equivalente nas classes de tabelas. Esta transformação dos dados brutos em um objeto facilmente manipulável ocorre ainda no âmbito dos DAOs.

Em cada um deles há um método específico para gerar um novo objeto de uma classe de tabela e fazer a separação dos dados recebidos do banco, inserindo-os no novo objeto criado.

As classes de tabelas também são utilizadas na inserção dos dados. Como os DAOs têm conhecimento da classe da tabela a qual se referem, eles também fazem, no momento da inserção no banco, a transformação dos dados do objeto em um elemento que será inserido no banco.

Os únicos métodos existentes em uma classe de tabela são os do tipo “get” e os do tipo “set”, uma vez que a função básica destas classes é servir de repositório temporário de dados.

2.6 – Classes de controle

Uma vez definido o acesso ao banco, garantindo seu isolamento do restante do sistema, é necessário que se implemente parte da regra de negócio do sistema. Mais

especificamente, a parte relativa aos dados que devem ser adquiridos, inseridos ou atualizados no banco.

Para isso são necessárias classes específicas de controle, que farão uso dos DAOs para fornecer ao restante do sistema os dados necessários na sua execução. Além disso, estas classes de controle usarão os DAOs para assegurar a persistência de dados vindos dos módulos WEB e de inteligência artificial.

Há também os métodos implícitos, que não passarão dados diretamente para os outros módulos, assim como não receberão informações diretamente deles. Executarão, assim, operações de forma implícita no BD, ou seja, executarão operações que não foram diretamente solicitadas por outro módulo, mas das quais alguma outra operação solicitada pode depender.

Ao contrário do que acontecia até este momento, as classes de controle não estão mais atreladas a uma tabela específica do banco e sim a um tipo específico de elemento a ser controlado.

A figura a seguir mostra a relação das quatro classes de controle com os DAOs.

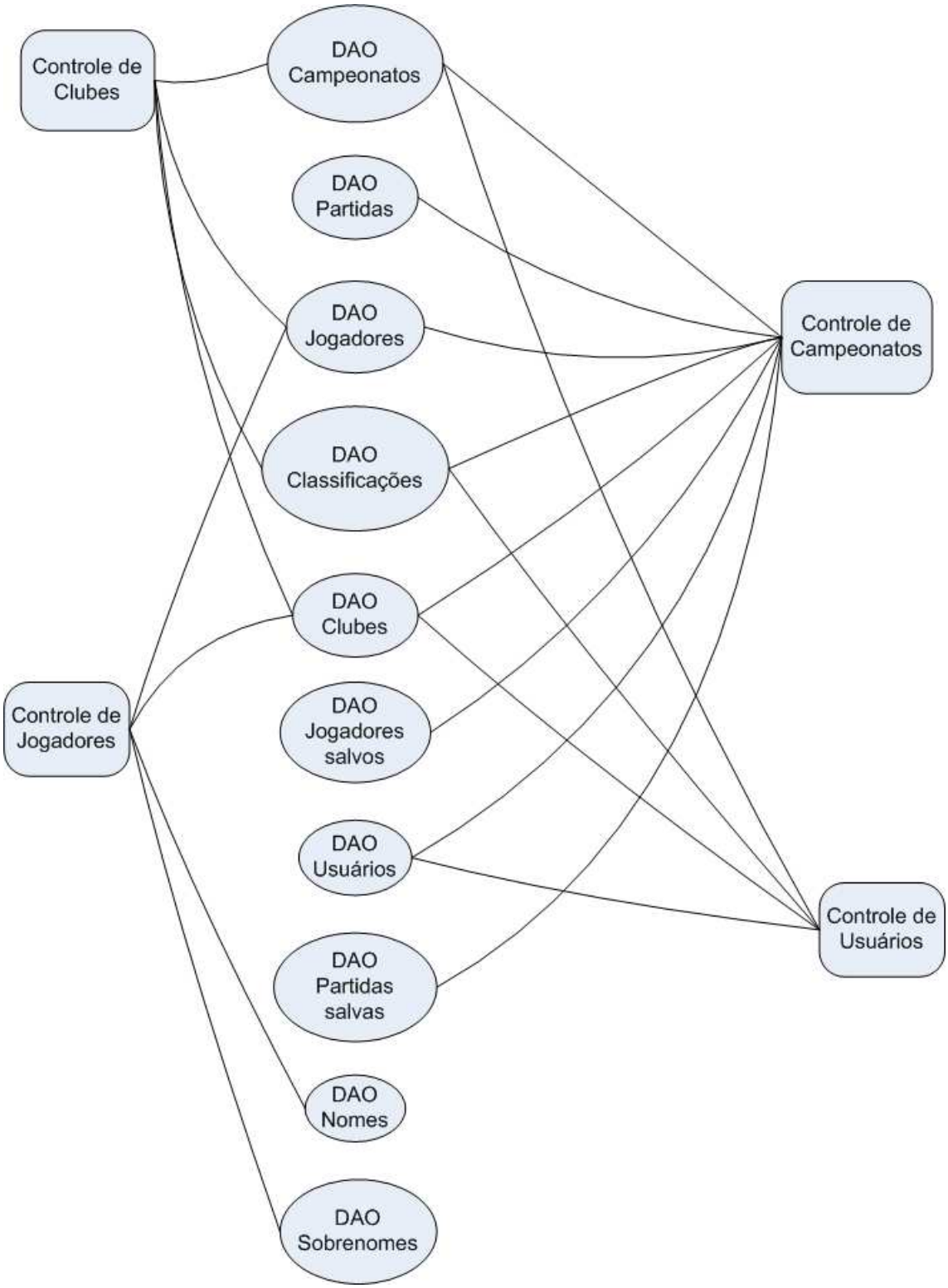

Figura 2.3 – Relações entre classes de controle e DAOs.

Logo abaixo serão detalhadas as funções destas classes.

2.6.1 – Controle de Campeonatos

Esta classe de controle serve para todo tipo de inserção, busca e alteração referentes a algum campeonato disputado através deste sistema. Abaixo estão identificados os seus métodos, acompanhados de breves descrições de suas funcionalidades.

Métodos explícitos:

1) Capturar nome do campeonato a partir de sua chave primária.
2) Mudar o nome de um determinado campeonato.
3) Capturar todos os campeonatos.
4) Capturar os campeonatos iniciados.
5) Configurar tempo restante para o início de um determinado campeonato.
6) Capturar o tempo existente entre duas rodadas.
7) Capturar a posição de um clube em um determinado campeonato.
8) Capturar o nome do criador de um determinado campeonato.
9) Capturar o número de times de um determinado campeonato.
10) Inserir um time em um determinado campeonato.
11) Capturar toda a classificação de um determinado campeonato.
12) Capturar o número total de rodadas de um determinado campeonato.
13) Capturar uma determinada rodada inteira de um determinado campeonato.
14) Executar uma determinada rodada de um determinado campeonato.
15) Capturar o número da última rodada executada.
16) Capturar o número da próxima rodada a executar.
17) Capturar as informações de uma determinada partida, de modo a poder visualizá-la.

Métodos implícitos:

1) Preencher lista com atributos dos jogadores. Usado na hora de executar uma rodada e de capturar informações de uma partida.
2) Inserir jogadores em um determinado time. Usado na hora de executar uma rodada.
3) Criar tabela de um determinado campeonato. Usado na hora de executar a primeira rodada.

2.6.2 – Controle de Clubes

Esta classe de controle serve para todo tipo de inserção, busca e alteração referentes a algum clube que esteja registrado neste sistema. Abaixo estão identificados os seus métodos, acompanhados de breves descrições de suas funcionalidades.

Métodos explícitos:

1) Criar um novo clube, tendo o nome do criador e o nome do clube.
2) Mudar o nome de um determinado clube.
3) Capturar os jogadores de um determinado clube.
4) Verificar se um determinado clube está em algum campeonato.

Não há métodos implícitos nesta classe de controle.

2.6.3 – Controle de Jogadores

Esta classe de controle serve para todo tipo de inserção, busca e alteração referentes a algum jogador de algum clube, que esteja registrado neste sistema. Abaixo estão identificados os seus métodos, acompanhados de breves descrições de suas funcionalidades.

Métodos explícitos:

1) Criar um novo jogador. Este método sorteia nome e sobrenome para o jogador, da lista de nomes e sobrenomes existentes no banco.
2) Trocar o valor do atributo de drible de um determinado jogador.
3) Incrementar o atributo de drible de um determinado jogador.
4) Decrementar o atributo de drible de um determinado jogador.
5) Trocar o valor do atributo de passe de um determinado jogador.
6) Incrementar o atributo de passe de um determinado jogador.
7) Decrementar o atributo de passe de um determinado jogador.
8) Trocar o valor do atributo de chute de um determinado jogador.
9) Incrementar o atributo de chute de um determinado jogador.
10) Decrementar o atributo de chute de um determinado jogador.

11) Trocar o valor do atributo de velocidade de um determinado jogador.

12) Incrementar o atributo de velocidade de um determinado jogador.

13) Decrementar o atributo de velocidade de um determinado jogador.

Métodos implícitos:

1) Modificar o atributo de drible de um determinado jogador. Este método é utilizado nos métodos explícitos de troca/incremento/decremento de drible.

2) Modificar o atributo de passe de um determinado jogador. Este método é utilizado nos métodos explícitos de troca/incremento/decremento de passe.

3) Modificar o atributo de chute de um determinado jogador. Este método é utilizado nos métodos explícitos de troca/incremento/decremento de chute.

4) Modificar o atributo de velocidade de um determinado jogador. Este método é utilizado nos métodos explícitos de troca/incremento/decremento de velocidade.

2.6.4 – Controle de Usuários

Esta classe de controle serve para todo tipo de inserção, busca e alteração referentes a algum usuário do sistema. Abaixo estão identificados os seus métodos, acompanhados de breves descrições de suas funcionalidades.

Métodos explícitos:

1) Criar novo usuário tendo seu login, senha e quantidade inicial de créditos.

2) Excluir um determinado usuário.

3) Mudar a senha de um determinado usuário.

4) Trocar o valor dos créditos de um determinado usuário.

5) Incrementar os créditos de um determinado usuário.

6) Decrementar os créditos de um determinado usuário.

7) Modificar login de um determinado usuário.

8) Verificar a senha de um determinado usuário.

9) Capturar os créditos de um determinado usuário.

10) Capturar a quantidade máxima de créditos que um determinado usuário pode comprar.

11) Verificar se um determinado usuário tem um clube em um determinado campeonato.

12) Capturar os campeonatos que foram criados por um determinado usuário.

13) Capturar os clubes que foram criados por um determinado usuário.

14) Criar um novo campeonato.

15) Capturar um determinado usuário através de seu login.

Método implícito:

1) Modificar créditos. Este método é usado pelos métodos explícitos de troca/incremento/decremento de créditos.

Capítulo 3
Interface WEB

3.1 – Descrição

Este capítulo tratará das estruturas utilizadas para se fazer a interface WEB para o usuário final do sistema. O objetivo desta interface é assegurar ao usuário uma boa navegabilidade, disposição agradável de elementos, além de permitir o isolamento de camadas do sistema.

A ideia central deste módulo é que seja possível alterá-lo, mudando, por exemplo, o idioma visível para o usuário, sem que se mexa no núcleo do código. Em outras palavras, a interface WEB corresponde à parte de visão do padrão MVC.

Neste capítulo serão abordados e explicados os conceitos de JSF, Facelets, CSS e Applets, suas aplicações, vantagens em relação a outras tecnologias e como seu uso se dá neste trabalho.

3.2 – JSF

JSF é um framework que permite a criação de páginas dinâmicas, usando o padrão MVC. Desta forma garante-se o isolamento de camadas e, consequentemente, o uso de boas práticas de programação.

A figura abaixo mostra a arquitetura JSF, baseada no modelo MVC.

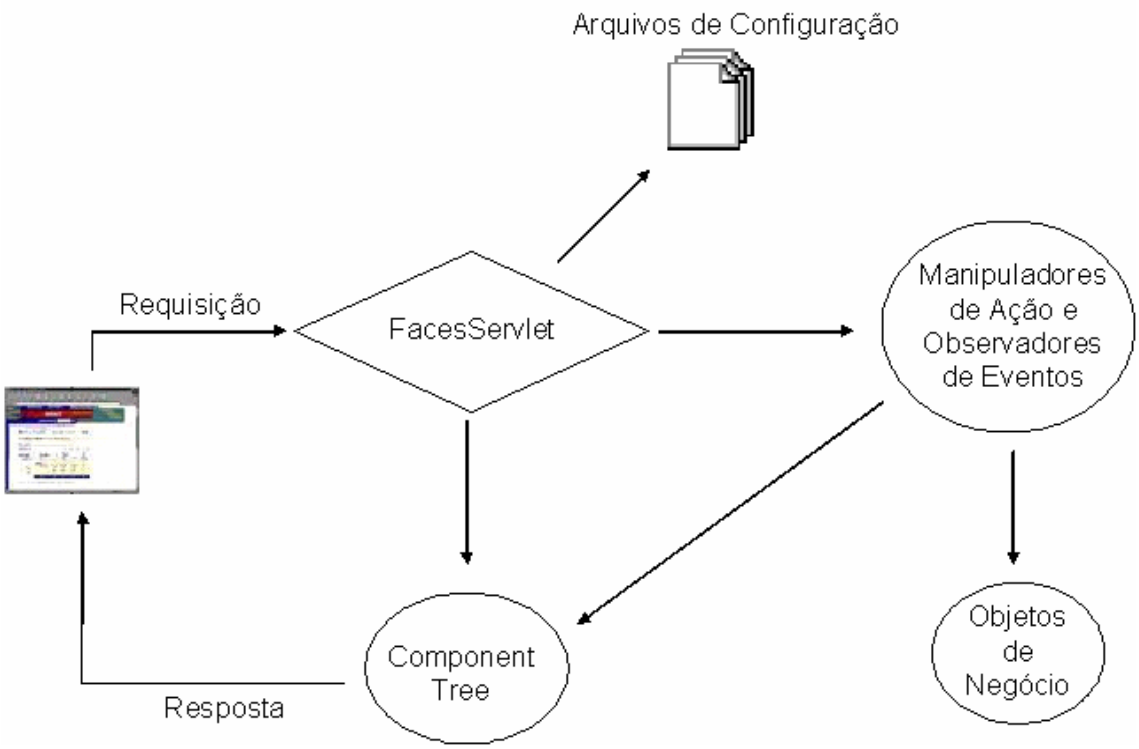

Figura 3.1 – Arquitetura JSF baseada no modelo MVC.

3.2.1 – Controle com JSF

O controle no JSF é composto de um servlet (o FacesServlet), manipuladores de ação e observadores de eventos e arquivos de configuração.

Um servlet é um pequeno programa, feito na linguagem Java, que encapsula uma funcionalidade da aplicação WEB. Sua execução é feita no lado do servidor. A ideia básica do servlet é receber requisições da parte do cliente e processá-los. Para este efeito, ele se comunica com os manipuladores de ação e os observadores de eventos, que se comunicam com os objetos de negócios, como pode ser visto na figura acima. São estes objetos que fazem consultas e/ou alterações no BD, como visto no capítulo 2. Estes objetos são a parte do modelo do JSF.

Uma vez que há o retorno de dados, ou o resultado da execução, por parte dos objetos de negócios, o FacesServlet prepara a parte que será visível para o usuário final, ou seja, prepara a parte de visão. Este resultado final pode ser representado, entre outras, por páginas HTML ou XML.

Os arquivos de configuração servem para estabelecer parâmetros para a parte de visão. Neste trabalho especificamente, os arquivos de configuração tem o objetivo de permitir a internacionalização, sendo sua implementação vista mais adiante.

3.2.2 – Visão com JSF

Esta parte do padrão MVC foi feita utilizando Facelets, applets e CSS.

Facelets é um templating framework, ou seja, é um subprojeto do JSF que tem o objetivo de facilitar, entre outras coisas, a construção da parte de visão do sistema. Até bem pouco tempo, a tecnologia mais utilizada para a criação de páginas dinâmicas era o JSP, que permitia uma forma menos elegante do padrão MVC. O uso de Facelets, ao contrário, permite o isolamento adequado de todas as camadas, a criação de componentes reutilizáveis, uma melhor reportagem de erros, além de ser consideravelmente mais rápido que o JSP, melhorando o desempenho do sistema. Outras características que favorecem o uso de Facelets são os fatos de ele ser independente do web container e de possuir conversão automática do HTML para componentes JSF, o que obviamente reduz o tempo gasto com a geração manual de código.

Applets nada mais são do que programas em Java que rodam no contexto de outro programa. De forma geral, executam operações bastante específicas. No caso particular deste projeto, eles rodam no web browser do usuário. Ou seja, no escopo deste sistema, os applets são softwares que rodam no lado do cliente. A utilização de applets aqui se dá na apresentação dos jogos que ocorrem entre os times dos usuários. Toda a visualização do transcorrer do jogo é feita em uma janela onde roda um applet, que utiliza toda a lógica da inteligência artificial, que será apresentada no capítulo 4.

Como visto no capítulo 2, há tabelas no BD que guardam informações sobre jogos que já tenham ocorrido. Estas informações permitem, a qualquer momento, a recuperação de um jogo e sua visualização fora do tempo de execução. Para que estes parâmetros sejam adequadamente utilizados pelo applet, eles devem vir da camada de controle do JSF. A ideia básica desta comunicação pode ser vista na figura abaixo.

Figura 3.2 – Transporte de dados entre o JSF e o Applet.

Como pode ser visto acima, o formato utilizado para o transporte de dados, do JSF para o applet, é o JSON. JSON nada mais é do que um formato de escrita de

estrutura de dados, utilizando-se de JavaScript. Falaremos inicialmente um pouco sobre JavaScript e, depois, entraremos no quesito JSON.

JavaScript é uma linguagem de programação, simples e interpretada. Sua utilização básica é executar pequenas ações, nos sistemas WEB, no lado do cliente, da mesma forma que o applet. A diferença básica entre eles é o fato de o applet, geralmente, contar com uma interface gráfica para o usuário, enquanto o JavaScript costuma apenas executar ações. Eventualmente conta com algum recurso visual.

Uma das estruturas nativas de JavaScript, JSON teve seu uso intensificado nos últimos tempos, em especial pelo fato de a leitura de dados neste formato ser mais rápida do que aquela feita a partir do XML. A lógica de estruturação de dados é semelhante à do XML, havendo pares "atributo-valor", porém de forma mais simplificada, o que obviamente facilita o acesso. Para a codificação destes parâmetros em JSON foi utilizada a biblioteca json2.js, em JavaScript.

Atributo	**Quantidade**
Dribbling	Um para cada jogador
Passing	Um para cada jogador
Shooting	Um para cada jogador
Velocity	Um para cada jogador
Nome do time	Um para cada time
Duração da partida	Única
Semente	Única

Tabela 3.1 – Atributos passados ao applet.

Outra tecnologia utilizada na camada de apresentação deste sistema é o CSS.

O CSS é uma linguagem para estilos que permite que se defina a aparência de uma página, separando o markup (HTML) da apresentação (CSS). A grande vantagem do uso desta tecnologia é justamente a separação de forma e conteúdo.

Inicialmente todas as páginas existentes na internet eram constituídas basicamente de HTML, uma linguagem de marcação. Assim, os elementos de uma página, como o cabeçalho e o corpo do hipertexto, por exemplo, eram definidas através de tags. Com o passar do tempo e a necessidade de se fazerem páginas com um visual melhor, criaram-se tags que não serviam exatamente à marcação de elementos. Foram criadas, por exemplo, tags que alterassem a cor ou o tipo da letra. Isso desvirtuou a ideia inicial do HTML e gerou a necessidade de novas ferramentas e tecnologias que permitissem a criação de um layout, que independesse da marcação do conteúdo da página.

A criação do CSS veio para resolver este problema. A estrutura básica de um código CSS é composta de três elementos, descritos abaixo:

1) Seletor: indica em qual tag HTML será aplicada a propriedade que se segue.

2) Propriedade: indica o tipo de característica para a qual se deseja usar o valor que se segue.

3) Valor: indica o valor da propriedade usada.

Desta forma, cada elemento de código CSS tem a estrutura dada abaixo:

Seletor {propriedade:valor}

A aplicação dos estilos das páginas deste sistema foi feita utilizando-se uma folha externa de estilos, como visto na figura abaixo. Esta folha consiste em um documento feito todo em código CSS, que define o layout desejado para as páginas. Em cada página do sistema há, então, um link para esta folha de estilos. Uma vez que ocorra uma alteração na folha de estilos, ela é automaticamente aplicada a todas as páginas do sistema. Podemos notar, então, que mesmo dentro de uma das camadas do sistema (no caso a camada de visão), há uma separação que permite um isolamento de subcamadas, diminuindo o retrabalho para a alteração de elementos reaproveitáveis.

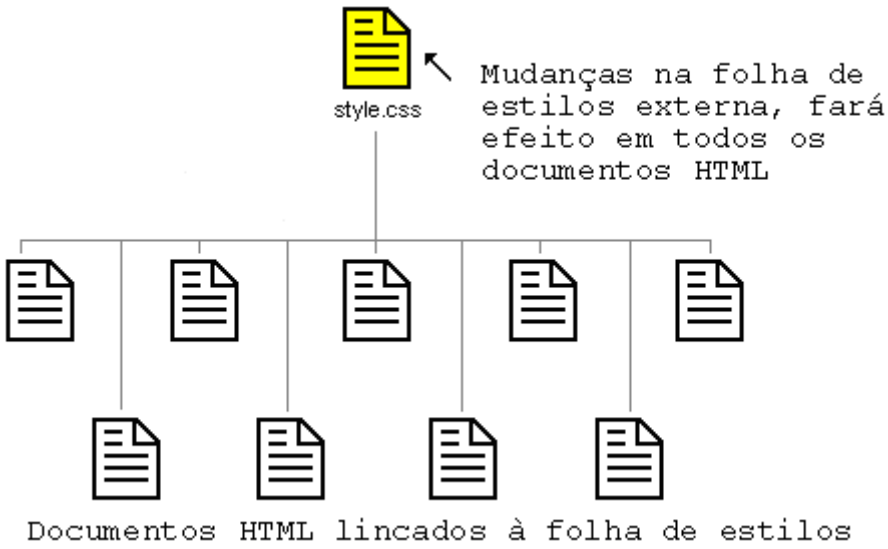

Figura 3.3 – Uso da folha de estilos externa.

3.2.3 – Implementação

Nesta seção abordaremos a implementação do JSF, cujas funcionalidades já foram abordadas anteriormente.

1) Recursos externos:

Aqui são colocadas frases e palavras para cada idioma.

Esta arquitetura permite internacionalização da aplicação. Todas as palavras e frases referentes a um idioma são colocadas em um único arquivo. Este arquivo é acessado em tempo de execução pelo servidor. O arquivo a ser acessado pelo servidor é aquele que contiver as definições referentes ao idioma selecionado.

Foram implementados os idiomas Inglês e Português. A estruturação dos dados referentes a idioma, dentro de cada arquivo, é feita da seguinte forma:

Arquivo de recursos para a língua inglesa:

window_title=Play right now !

sys_desc_1=Multiuser Interactive Web Platform Entertainment Environment Integrated with Artificial Intelligence Expert System.

sys_desc_2=This game was designed to be an online soccer club management simulator.

sys_desc_3=Create your soccer club and go compete with other players in our leagues.

sys_desc_4=Show your habilities and take the throphy!

login_label=Play right now

username_label=Login

password_label=Password

login_btn_label=Enter

create_account_img=images/create_account_en.png

create_account_label=Create your account

register_account_label=Register

back_label=back

...

Arquivo de recursos para a língua portuguesa:

window_title=Jogue agora mesmo!

sys_desc_1=Sistema Especialista de Inteligencia artificial Integrado a Plataforma Web para Entretenimento em um Ambiente Interativo Multiusuário.

sys_desc_2=Este jogo foi projetado para ser um simulador online de gerenciamento de clubes de futebol.

sys_desc_3=Aqui voce cria o seu time e compete com outros jogadores em diversos campeonatos.

sys_desc_4=Mostre suas habilidades e seja campeão!

login_label=Jogue agora mesmo

username_label=Usuário

password_label=Senha

login_btn_label=Entrar

create_account_img=images/create_account_pt.png

create_account_label=Crie sua conta

register_account_label=Registrar

back_label=voltar

...

Com essa estrutura simples é feito o suporte à internacionalização. A grande vantagem é que desta forma a base de código da aplicação não é afetada por questões relativas à seleção de idioma. Além disso, esta arquitetura permite que pessoas diferentes trabalhem em paralelo, promovendo uma separação de camadas elegante e funcional. Para adicionar um dado idioma todo o necessário é criar um novo arquivo contendo as definições específicas daquele idioma.

2) Controle de navegação:

Este módulo contém a implementação de todas as interconexões existentes entre as páginas.

As transições entre páginas ocorrem em função do estado da aplicação.

A seguinte figura contém um fragmento que ilustra algumas das conexões existentes:

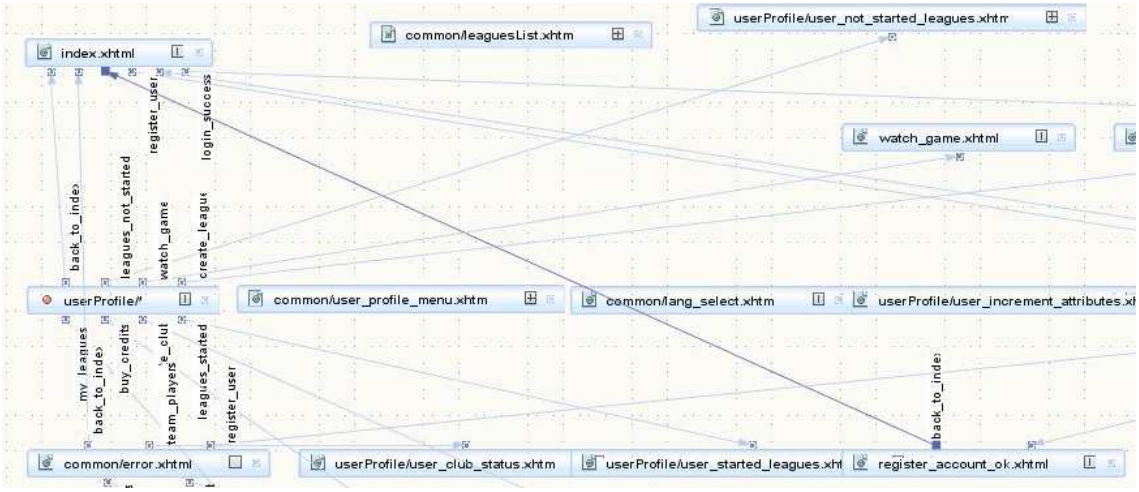

Figura 3.4 – Parte da captura de tela da representação de conexões existentes.

Pode-se perceber, por exemplo, que a página index.xhtml se comunica bilateralmente com as páginas userProfile/* .

Toda a regra de negócio que delimita a interconexão entre as páginas é definida em um arquivo XML chamado faces-config.xml.

Um trecho deste arquivo:

....

```
<navigation-rule>
  <from-view-id>/index.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>login_success</from-outcome>
    <to-view-id>/userProfile/user_instructions.xhtml</to-view-id>
    <redirect/>
  </navigation-case>
  <navigation-case>
    <from-outcome>register_user</from-outcome>
    <to-view-id>/register_account.xhtml</to-view-id>
    <redirect/>
  </navigation-case>
</navigation-rule>
```

....

Este trecho ilustra que a página index.xhtml possuí duas transições:

I) Login bem sucedido: vai para a página /userProfile/user_instructions.xhtml

II) Registro de usuário: vai para a página /register_account.xhtml

3) Interface com o usuário:

Como dito anteriormente, esta parte da aplicação foi construída utilizando-se a arquitetura de facelets.

Compõe a apresentação da aplicação. É diretamente através dela que o usuário se comunica com o sistema.

Estrutura da página index.xhtml:

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:f="http://java.sun.com/jsf/core">

  ${userSession.setLocale()}

  <h:head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>#{msgs.window_title}</title>
    <link href="css/main.css" rel="stylesheet" type="text/css" />
    <link href="css/common.css" rel="stylesheet" type="text/css" />
  </h:head>
  <h:body>
    <ui:include src="common/header.xhtml"/>
    <ui:include src="common/lang_select.xhtml"/>
    <div id="page_content">
      <h:panelGrid columns="3">
        <h:panelGroup id="project_info" layout="block">
          <p id="project_name">
            <h:outputText value="#{msgs.sys_desc_1}"/>
          </p>
          <p><h:outputText value="#{msgs.sys_desc_2}"/></p>
          <p><h:outputText value="#{msgs.sys_desc_3}"/></p>
          <p><h:outputText value="#{msgs.sys_desc_4}"/></p>
        </h:panelGroup>

        <h:panelGroup id="main_page_center_column" layout="block"/>

        <h:panelGroup styleClass="box_login_frame" layout="block">
          <p class="font_header center_text" id="play_now">
```

```
    <h:outputText value="#{msgs.login_label}"/>
</p>
<h:form>
    <h:panelGrid styleClass="login_table" columns="3">

        <h:panelGroup styleClass="login_table_label_column" layout="block">
            <h:outputLabel  for="login_input_name"
                            styleClass="bold_text"
                            value="#{msgs.username_label}:"/>
        </h:panelGroup>

        <h:panelGroup>
            <h:inputText label="login" id="login_input_name" styleClass="login_input"
                         value="#{login.loginName}" required="true"
                         requiredMessage="#{msgs.field_required}"/>
        </h:panelGroup>
        <h:message for="login_input_name" styleClass="validation_error"/>

        <h:panelGroup styleClass="login_table_label_column" layout="block">
            <h:outputLabel for="login_input_password"
                           styleClass="bold_text"
                           value="#{msgs.password_label}:"/>
        </h:panelGroup>

        <h:panelGroup>
            <h:inputSecret label="password" id="login_input_password"
                           styleClass="login_input"
                           value="#{login.password}"
                           required="true"
                           requiredMessage="#{msgs.field_required}"/>
        </h:panelGroup>
        <h:message for="login_input_password" styleClass="validation_error"/>

    </h:panelGrid>

    <div id="login_btn_id" class="center_text">
        <h:commandButton value="#{msgs.login_btn_label}"
                         action="#{login.CheckValidUser()}"/>
    </div>
```

```
            </h:form>

            <h:form id="create_account_btn">
               <h:commandLink  action="#{nav.gotoRegister()}">
                  <h:graphicImage styleClass="center_image"
                              value="#{msgs.create_account_img}"
                              alt="create_acount"/>
               </h:commandLink>
            </h:form>
         </h:panelGroup>  <!--end box login frame-->
      </h:panelGrid>
   </div> <!-- end page_content -->

   <ui:include src="common/footer.xhtml"/>
  </h:body>
</html>
```

Pode-se perceber por inspeção que a página contém "tags" especiais. Estas tags são definidas pela especificação JSF.

Como citado anteriormente, o servlet traduz essas tags em tempo de execução para tags HTML.

Este trecho exemplifica o funcionamento do formulário de login:

```
<h:form>

        <h:panelGrid styleClass="login_table" columns="3">
...

        <h:panelGroup>
              <h:inputText label="login" id="login_input_name" styleClass="login_input"
            value="#{login.loginName}" required="true"
            requiredMessage="#{msgs.field_required}"/>
  </h:panelGroup>
...

  <div id="login_btn_id" class="center_text">
        <h:commandButton value="#{msgs.login_btn_label}"
                                    action="#{login.CheckValidUser()}"/>
  </div>
 </h:form>
```

Ao se clicar no botão (tag h:commandButton) o servlet que controla o JSF redireciona o controle para o método login.CheckValidUser().

Este método, por sua vez, chama o método de autentição do WS passando como parâmetros o nome de usuário e a senha contida em suas variáveis membro. Os "Beans de Sessão" (descritos no próximo sub-item) são responsáveis por guardar os valores fornecidos pelos usuários.

Abaixo a tela index.xhtml exibida pelo navegador:

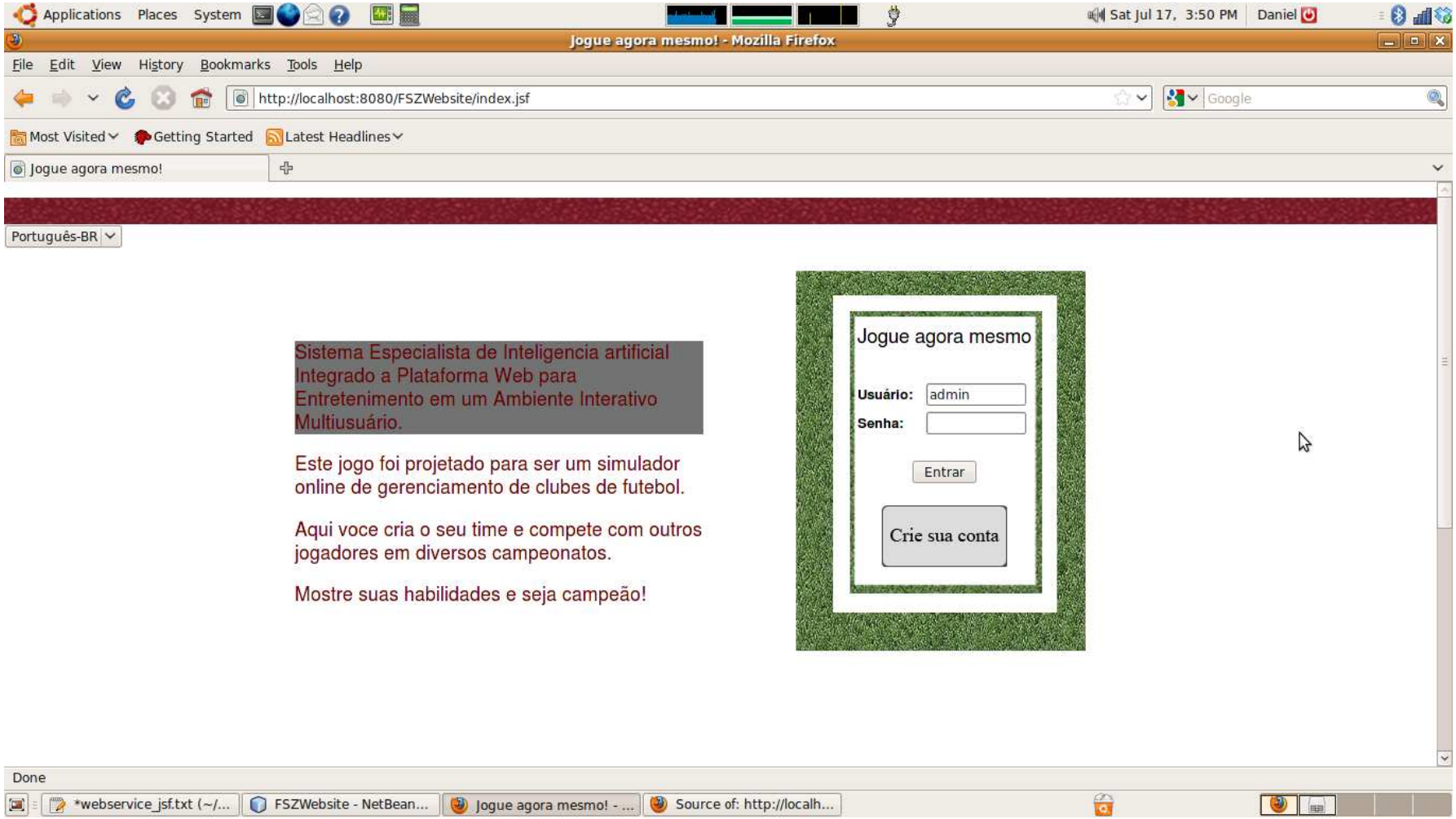

Figura 3.5 – Tela index.xhtml.

Tela que exibe o andamento de um campeonato:

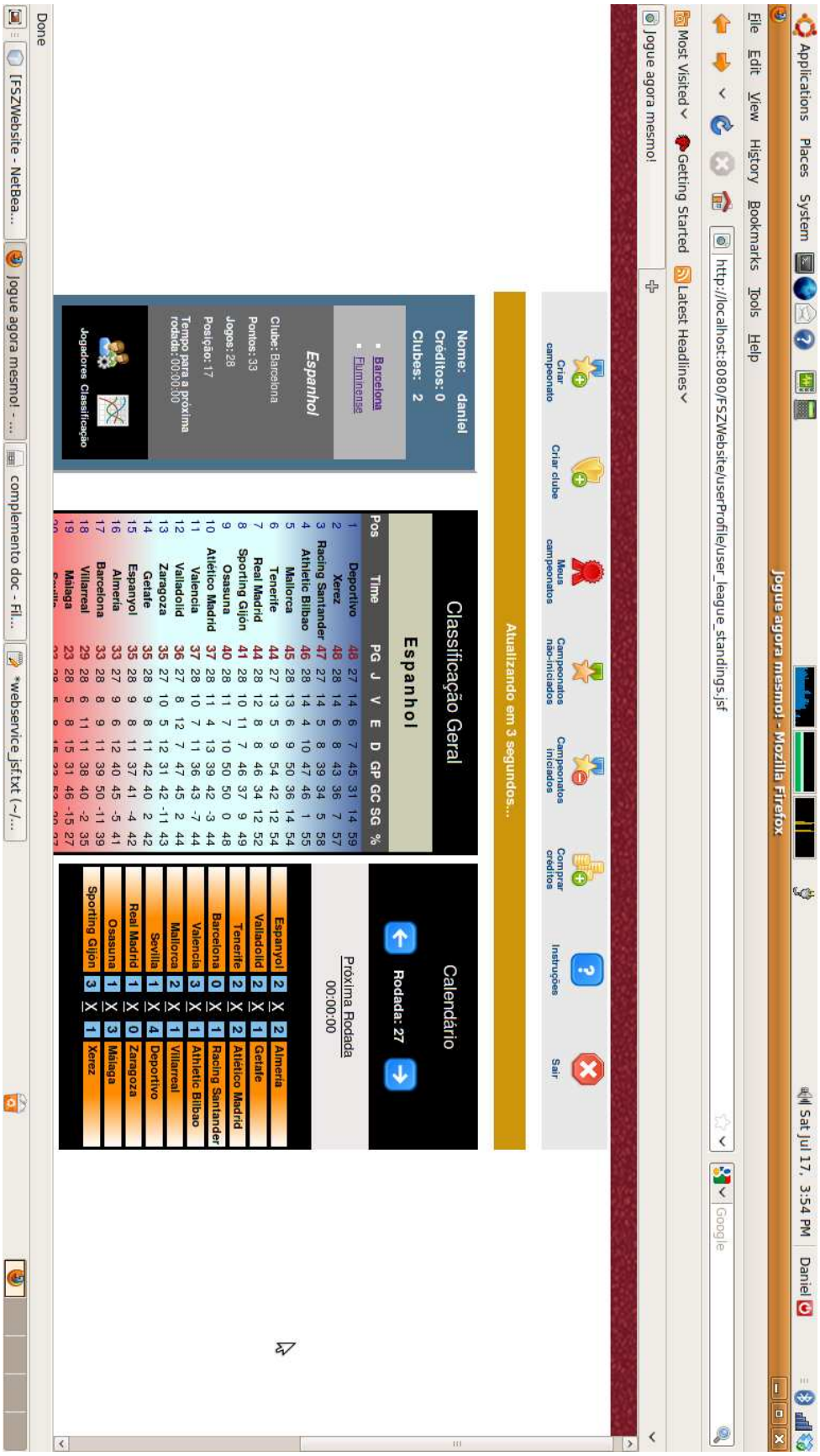

Figura 3.6 – Andamento de um campeonato.

4) Beans de Sessão:

Estes são objetos java responsáveis por receber os dados vindos da interface gráfica e fornecê-los ao controle da aplicação.

O Bean responsável pelo Login é descrito abaixo:

...

```
public class LoginBean extends UserBean{

   private String m_loginName = AppDefinitions.ADMIN_USERNAME;
   private String m_password;

   public void setLoginName(String loginName) {
      m_loginName = loginName;
   }

   public String getLoginName() {
      return m_loginName;
   }

   public void setPassword(String password) {
      m_password = password;
   }

   public String getPassword() {
      return m_password;
   }

   public String CheckValidUser() {
      WsOperationContext operationCtx = FSZWSClientStub.doLogin(m_loginName, m_password);

      String action;

      if (operationCtx.isOK()) {
         UserSessionManager.resetAppState();
         action = JSFNavigation.gotoLoginSuccess();
      } else {
         action = JSFNavigation.gotoErrorPage(AppDefinitions.MESSAGE_BUNDLE,
```

```
                                AppDefinitions.BUNDLE_MSG_ERROR_AUTHENTICATION,
                                JSFNavigation.gotoIndex());
        }
        return action;
    }

    @Override
    public void resetBean() {
        m_loginName = AppDefinitions.ADMIN_USERNAME;
        m_password  = null;
    }
}
```

Os métodos getLoginName e getPassword fornecem à interface os dados provindos do estado atual do sistema.

Já os métodos setLoginName e setPassword fazem a entrada de dados vindos do usuário através da interface.

O método CheckValidUser chama o método do WS responsável pela autenticação e retorna o status da operação.

Dependendo do status o sistema faz login ou retorna uma mensagem de erro.

Em caso de sucesso o usuário passa para área restrita e se depara com a tela de instrução.

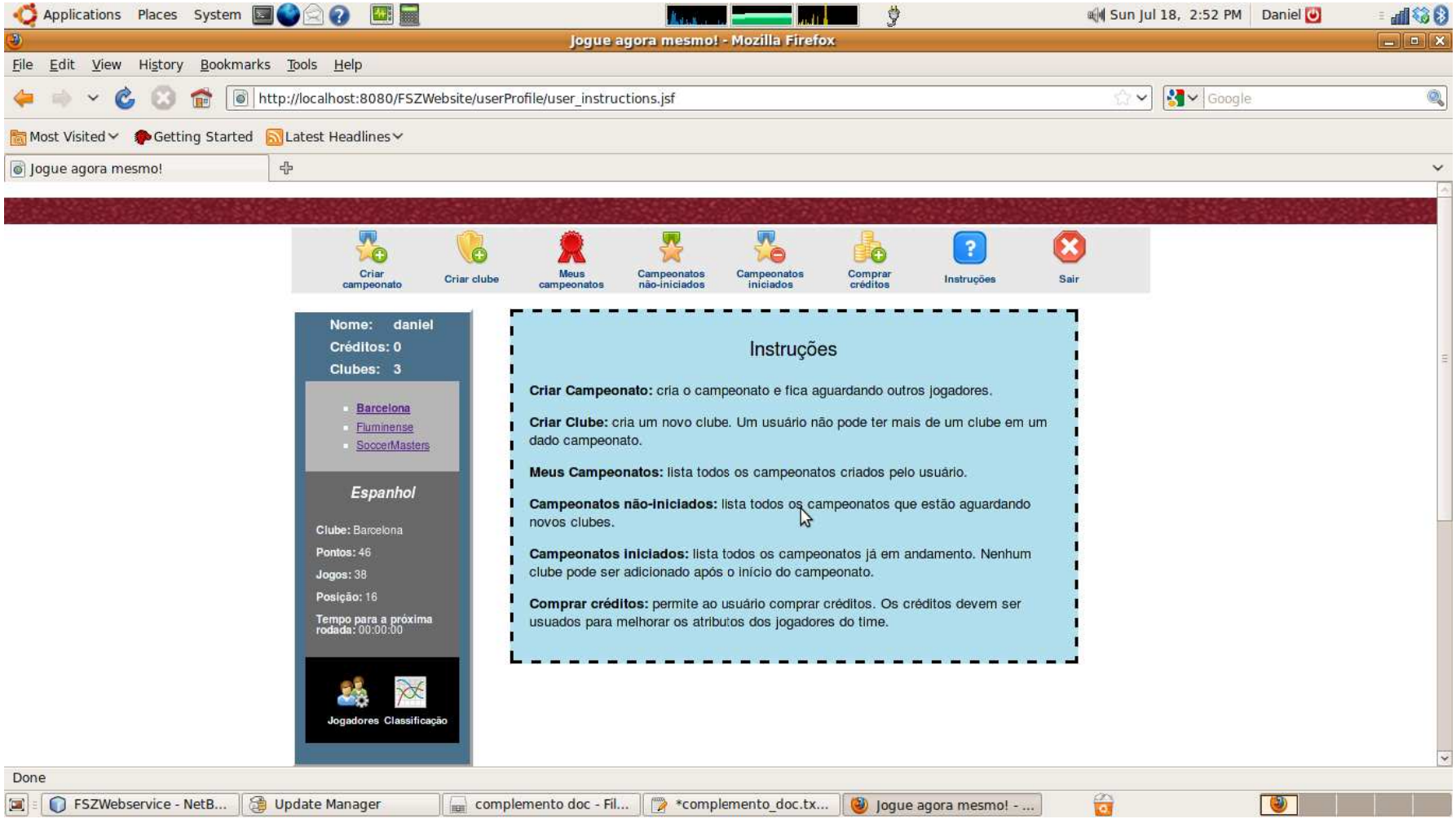

Figura 3.7 – Tela de instruções.

Em caso de erro a mensagem apropriada é exibida ao usuário, por exemplo, erro de autenticação:

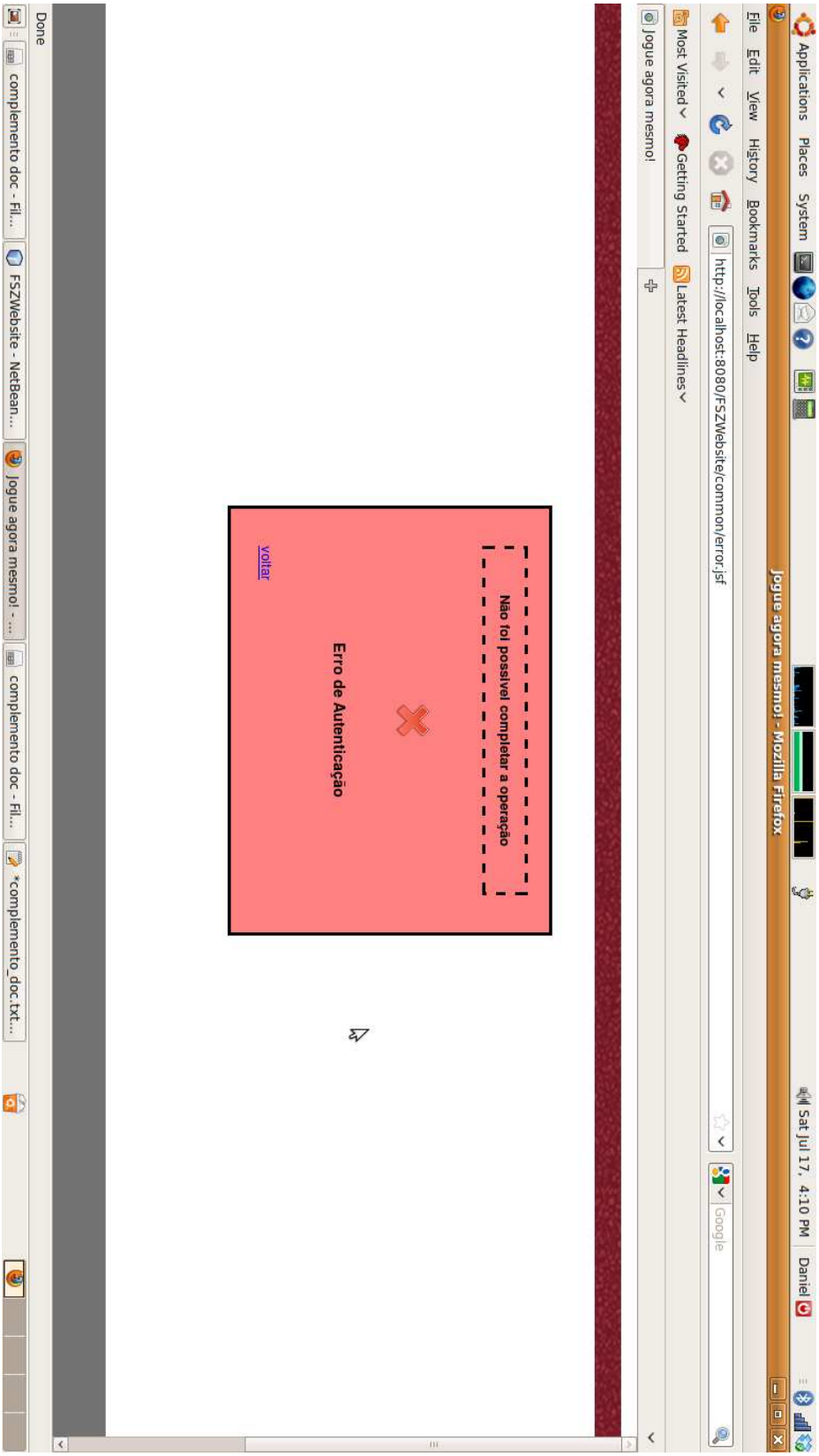

Figura 3.8 – Tela de erro de autenticação.

Capítulo 4
Inteligência Artificial

4.1 – Descrição

Neste capítulo será demonstrado como cada tática e movimento específico deve ser modelado e serão apresentados os conceitos básicos necessários para se implementar computacionalmente um time de futebol. O ambiente de jogo foi criado de forma bastante simples e algumas regras do esporte considerado (futebol) foram desprezadas para que fosse praticável a implementação de um protótipo em um projeto final de curso.

Existem dois times: azul e vermelho. Cada time possuí quatro jogadores e um goleiro. Por razões de simplicidade o campo de futebol considerado possuí “paredes” em suas laterais e portanto a bola nunca sai de campo, sempre quica na parede e volta para o campo de jogo. Portanto não existe lateral ou escanteio. Outras situações como impedimento e marcação de faltas também foram desconsideradas.

Segue abaixo o diagrama UML das classes que compõem o módulo de IA do projeto.

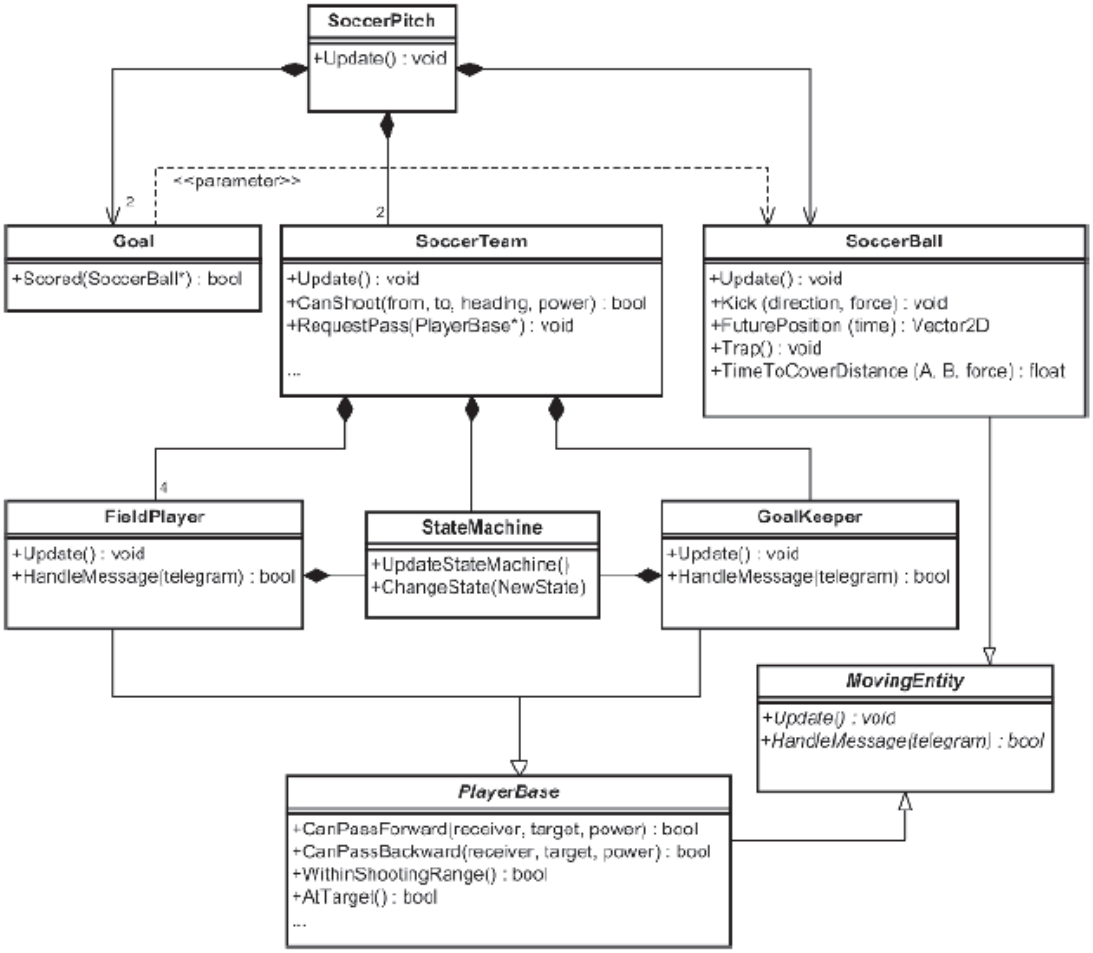

Figura 4.1 – Diagrama UML das classes do módulo de IA.

4.2 – Definição

Aqui será definido cada elemento componente do motor de Inteligência Artificial e ao final desta seção será apresentado um diagrama UML contendo o relacionamento entre as entidades consideradas.

O ambiente de jogo consiste dos seguintes itens:

4.2.1 - Campo de futebol

O campo de futebol é uma área retangular envolvida por paredes em todo o seu perímetro. Existe um gol de cada lado posicionado de maneira centralizada. A bola é posicionada no centro do campo antes do início da partida e após a marcação de um gol.

A classe “SoccerPitch.java” representa o campo de futebol e possuí os seguintes membros:

4.2.1.1 – Bola de futebol

Este elemento encapsula a bola de futebol e possuí todos os métodos necessários para reproduzir a física de seu movimento e interação com o meio físico. Está implementada na classe SoccerBall presente arquivo SoccerBall.java.

4.2.1.2 – Times de futebol

Aqui estão presentes os times de futebol. Cada time é representado por um membro da classe SoccerTeam implementada no arquivo SoccerTeam.java.

4.2.1.3 – Gols

Aqui estão presentes os gols a serem defendidos (e atacados) pelos jogadores em campo. Este elemento serve como agente delimitador da área que define a região em que a bola deve adentrar para que seja marcado gol e a partida seja reiniciada. A implementação deste elemente se encontra na classe Goal presente no arquivo Goal.java.

4.2.1.4 – Paredes laterais

Este elemento define o perímetro da região de jogo. Aqui estão definidos os vetores normais que determinam os ângulos de reflexão da bola em caso de colisão. A implementação deste elemento se encontra na classe Wall2D presente no arquivo Wall2D.java.

4.2.1.5 – Regiões

O campo de futebol é dividido em dezoito regiões conforme a figura a seguir.

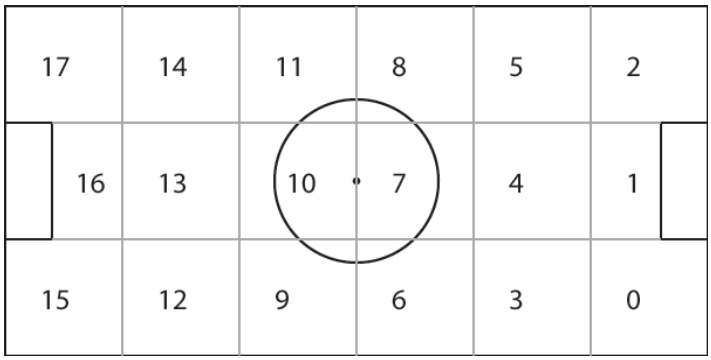

Figura 4.2 – Regiões do campo de futebol.

A utilidade de se dividir o campo em diversas regiões é que as mesmas podem ser utilizadas como bases de atuação para determinados jogadores exercendo determinadas funções, por exemplo atacantes devem atuar em regiões adiantadas enquanto zagueiros devem se manter atrás, nas regiões correspondentes ao campo de defesa. Além disso as regiões podem ser utilizadas para se definir estratégias e esquemas táticos.

Cada jogador possui uma região base para atuar e devem retornar às mesmas sempre que o jogo for reiniciado.

4.3 - Projeto do motor de Inteligência Artificial

Nesta seção serão descritos os algoritmos utilizados para gerar a coordenação dos elementos componentes do jogo. Será explicado o funcionamento de uma máquina de estado finita e de que forma a mesma foi utilizada para modelar as mais diversas ações que podem ser tomadas por um jogador durante o curso da partida.

4.3.1 – Máquina de Estados Finita (MEF)

As máquinas de estado finitas desempenham papel fundamental na indústria dos jogos eletrônicos. Estas máquinas são há muitos anos o instrumento de escolha dos programadores de Inteligência Artificial para criar a ilusão de inteligência em um agente programado.

As vantagens de se utilizar MEF é que as mesmas são simples de ser implementadas e simples de ser analisadas tendo em vista que uma vez que o comportamento de um agente pode ser quebrado em diversos estados se torna mais fácil descobrir em que estado está ocorrendo um possível erro de programação.

Além disso, podemos citar que MEF consomem pouquíssimos recursos de CPU, são intuitivas ao reproduzir a tendência humana de dividir atividades em categorias e flexíveis uma vez que é teoricamente simples aprimorar um determinado aspecto de um estado, ou um relacionamento entre dois ou mais estados, ou ainda criar novas transições.

Podemos definir formalmente uma MEF da seguinte forma:

"Uma Máquina de Estado Finita é um dispositivo ou modelo de dispositivo, o qual possuí um número finito de estados e pode reagir a uma determinada entrada produzindo uma transição entre estados ou causando a ocorrência de uma determinada saída. Uma MEF deve estar em um e somente um estado de cada vez."

4.3.1.1 – Projeto da Máquina de Estados

Na elaboração do jogo foram projetados dois níveis de máquinas de estados. Cada time corresponde a uma máquina de estados e cada jogador de um time também corresponde a uma máquina de estados. Esta arquitetura separada em camadas torna independente o código utilizado para coordenar a entidade jogador do código utilizado para coordenar a entidade time.

Vamos considerar separadamente as transições de estado pertencentes às entidades "time", "jogador de campo" e "goleiro".

As seções abaixo explicarão o funcionamento da máquina de estados para cada entidade.

4.3.1.1.1 – Diagrama de Estados da Entidade Time

A entidade time é constituída por três estados (Figura 4.2).

As setas representam as transições possíveis para cada estado. As transições ocorrem se o time perder a posse de bola, marcar um gol ou sofrer um gol, conforme indicado no texto sobre as setas.

Abaixo uma descrição de cada estado:

Atacar: Neste estado o time possuí a bola e os jogadores se posicionam mais a frente no campo de jogo.

Defender: Neste estado o time não possuí a posse de bola e os jogadores se posicionam mais atrás no campo de jogo.

Disputar saída de bola: Este estado corresponde ao estado dos times antes do início da partida ou após a marcação de um gol. Todos os jogadores encontram-se em suas posições de origem e disputam a bola no meio de campo.

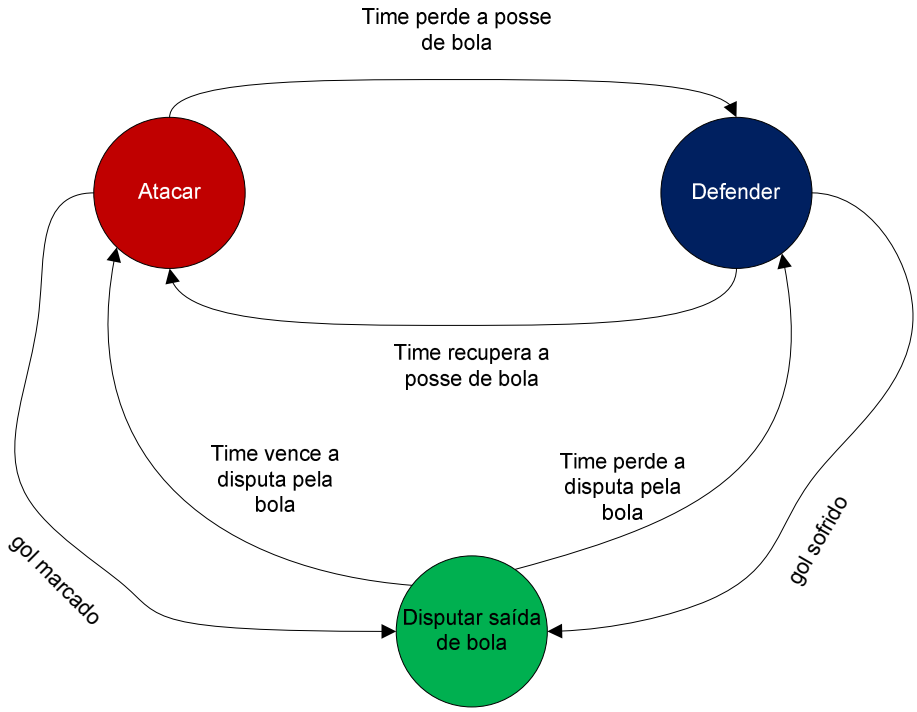

Figura 4.3 – Diagrama de estados da entidade Time.

4.3.1.1.2 – Diagrama de Estados da Entidade Jogador de Campo

A entidade jogador é composta por oito estados (Figura 4.3).

As setas representam as transições possíveis para cada estado.

Todo jogador possuí simultâneamente duas máquinas de estado.

Uma que realiza transições de estado devido à mensagens recebidas de outros jogadores (Máquina de Estados Global) e outra que realiza transições devido à ocorrência de eventos como por exemplo a perda da posse de bola ou a marcação de um gol.

Máquina de Estados Global:

Esta máquina de estados é responsável por fazer as transições de estado devido ao recebimento de mensagens vindas de outros jogadores. Um exemplo do

funcionamento do “estado global” é quando um jogador se encontra em uma boa posição para receber um passe e envia uma mensagem ao “estado global” do jogador que possuí a posse da bola. O estado global do jogador com a bola recebe a mensagem, processa e, se for o caso, envia uma mensagem de volta com as coordenadas de onde a bola será passada. O Estado global do receptor recebe esta mensagem, e processa a mudança para o estado “Receber a bola”.

Existem cinco mensagens utilizadas:

- Acompanhe o Ataque:

Informa ao estado global do destinatário que o jogador com a bola está precisando de outro para ajudar na execução do ataque.

- Retorne à posição Inicial:

Ordena ao jogador para ele retornar à sua posição de origem.

- Receba a Bola:

Informa ao jogador alvo que lhe será feito um passe. As coordenadas do passe sendo executado são informadas.

- Passe para mim:

Se um determinado jogador se encontrar em uma posição vantajosa ele pode solicitar ao jogador com a posse da bola que lhe seja feito o passe.

- Espere:

Ordena ao jogador para ele permanecer em sua posição.

Abaixo uma descrição de cada estado:

Espera:

Neste estado o jogador irá permanecer em sua posição atual. Existem duas possibilidades para se sair deste estado. Se o jogador for o mais próximo da bola e seu time esteja sem a posse de bola ou se o jogador estiver em uma posição vantajosa para receber o passe. Neste último caso o jogador irá enviar uma mensagem ao jogador com a posse de bola requisitando que o passe seja feito.

Receber a bola:

Um jogador entra neste estado quando processa a mensagem “Receba a Bola”. Esta mensagem é enviada ao receptor pelo jogador que acabou de fazer o passe. Deve haver apenas um jogador do time neste estado. Não seria uma boa tática ter dois ou mais

jogadores tentando interceptar o mesmo passe. O receptor irá mover-se para a posição e ficará por lá até receber o passe ou seu time perder a bola. Em qualquer das situações ocorrerá a transição para o estado “Perseguir a bola”.

Perseguir a bola:

Neste estado o jogador irá perseguir a bola até que a mesa este em seu alcance para ser chutada. Se o jogador deixar de ser o mais próximo da bola, o mesmo irá retornar à sua posição inicial.

Chutar a Bola:

Este estado implementa a lógica necessária para chutar a gol e passar a bola. Se por algum motivo um jogador não pode chutar à gol e um passe não é necessário ocorrerá uma transição para o estado “Driblar”.

Se um chute a gol não é possível o jogador considera fazer um passe. Um jogador irá considerar esta possibilidade apenas se estiver ameaçado por outro jogador. Antes de fazer o passe o jogador analisa se existe um caminho livre entre ele o receptor. Se for seguro fazer o passe o jogador envia a mensagem “Receba a Bola” para o estado global do receptor.

Uma vez que a bola seja chutada ocorre uma transição para o estado “Espera” e é enviada uma mensagem solicitando que algum outro jogador se apresente para participar da jogada.

Driblar:

Neste estado o jogador conduz a bola pelo campo de jogo através de uma série de pequenos chutes. Usando esta técnica um jogador é capaz de rotacionar além de contornar o corpo de um adversário enquanto possuí a bola. Sempre ocorre uma transição para o estado “Perseguir a bola” após o drible.

Retornar à posição inicial:

O jogador entra neste estado ao receber a mensagem “Retorne à posição inicial” e permanece no mesmo até que esteja em sua posição de origem onde fica até que uma das seguintes situações ocorra:

- Seu time está com a bola e o jogador neste estado se encontra à frente do jogador que conduz a bola. É feita uma requisição para que o passe seja executado através da mensagem “Passe para mim”. Caso a mensagem seja aceita pelo jogador com a bola o mesmo enviará a mensagem “Receba a bola” para o estado global do jogador que fez a solicitação de passe. Esta mensagem fará com que ocorra uma transição para o estado “Receber a bola”.

- É o jogador mais próximo da bola e não existe um receptor alvo de seu time naquele instante. Além disso o goleiro não está com a posse de bola. Neste caso ocorrerá uma transição para o estado “Perseguir a bola”.

Acompanhar o ataque:

Quando um jogador obtém o controle da bola ele imediatamente pede ajuda a outro jogador do time para acompanhar sua jogada. O jogador com a bola envia a mensagem “Acompanhe o Ataque” ao jogador alvo. Este jogador, então, se encaminha para a “posição de suporte” indicada na mensagem.

Se o time perder a posse da bola o jogador neste estado retorna imediatamente à sua posição inicial.

Ao chegar na “posição de suporte” o jogador neste estado envia uma mensagem ao jogador com a bola solicitando que o passe seja feito (Mensagem “Passe para mim”).

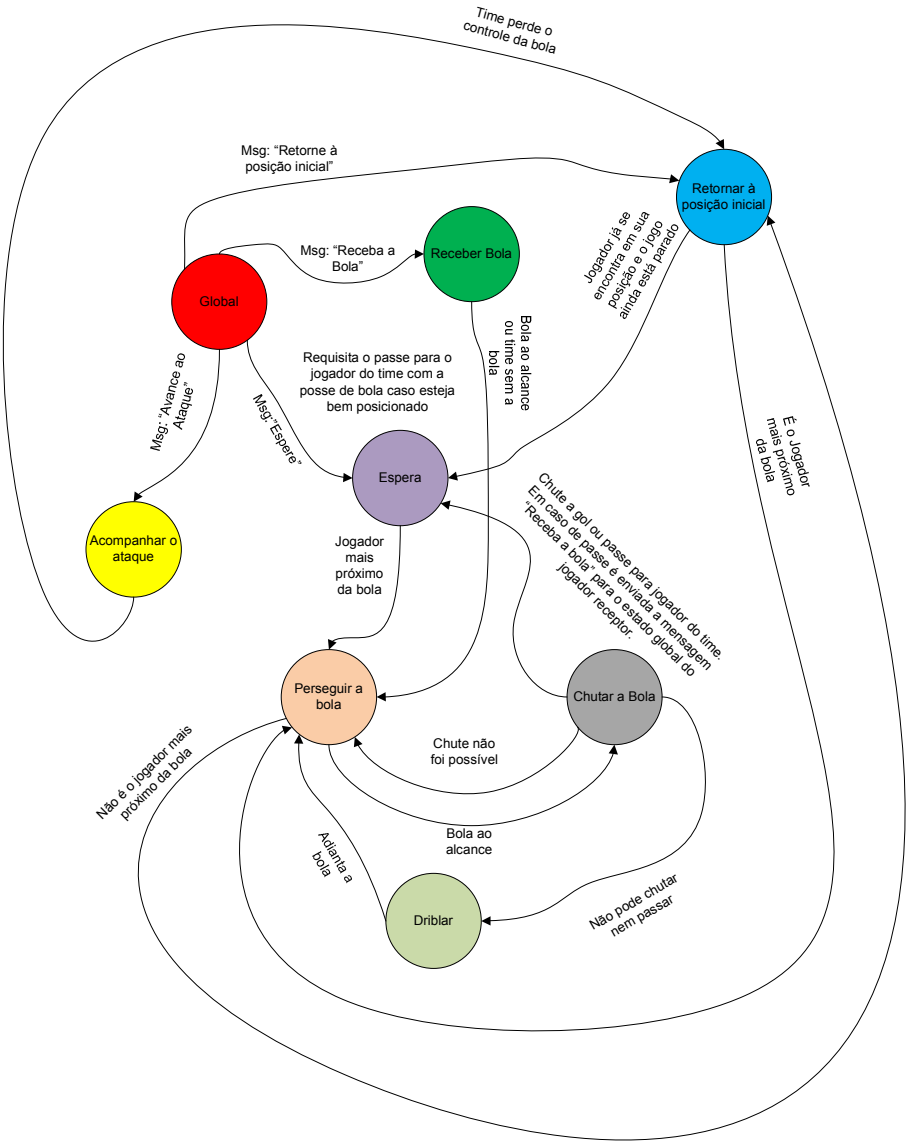

Figura 4.4 – Diagrama de estados da entidade jogador de Campo.

4.3.1.1.3 – Diagrama de estados da entidade Goleiro

A entidade goleiro é composta por cinco estados (Figura 4.4).
As setas representam as transições possíveis para cada estado.

Todo goleiro possuí simultâneamente duas máquinas de estado.

Uma que realiza transições de estado devido à mensagens recebidas de outros jogadores (Máquina de Estados Global) e outra que realiza transições devido à ocorrência de eventos como por exemplo a perda da posse de bola ou a marcação de um gol.

Máquina de Estados Global:

Assim como na entidade jogador de campo, a máquina de estados Global serve como um "roteador" para as mensagens que a entidade goleiro pode receber. Um goleiro recebe apenas dois tipos de mensagens :

Retorne à posição Inicial:

Ordena ao goleiro para ele retornar à sua posição de origem.

- Receba a Bola

Informa ao goleiro que a bola está sendo passada para ele.

Abaixo uma descrição de cada estado:

Defender o gol:

Neste estado o goleiro se posiciona embaixo das traves e se prepara para pular em direção à trajetória da bola. O Goleiro se move lateralmente sob o gol tentando manter seu corpo entre a bola e a linha imaginária que define a trajetória do chute. O goleiro permanece neste estado até que ocorram duas possíveis situações:

1) A bola se encontre em uma distância em que pode ser disputada com o jogador adversário fazendo com que haja uma transição para o estado "Interceptar a bola"
2) A bola se encontre sob seu domínio fazendo com que haja uma transição para o estado "Recolocar a bola em jogo"

Retornar à posição inicial:

Neste estado o goleiro retorna para a região onde se localiza o gol a ser defendido. Caso seu time perca a posse de bola goleiro fará uma transição para o estado “Defender o gol”.

Recolocar a bola em jogo:

Quando o goleiro ganha a posse de bola ele entra no estado “Recolocar a bola em jogo”. Ao entrar neste estado o goleiro avisa aos jogadores para retornarem às suas posições iniciais para que o “tiro de meta” possa ser batido. Quando estiverem em posição o goleiro passará a bola para um jogador de seu time e logo em seguida fará uma transição para o estado “Defender o gol”.

Interceptar a bola:

O goleiro tentará interceptar a bola quando o oponente se aproximar perigosamente de seu gol. Neste estado o goleiro avança em direção ao jogador adversário para tentar roubar a bola. Caso o goleiro recupere a posse de bola ele faz uma transição para o estado “Recolocar a bola em jogo”. Caso o goleiro julgue estar muito longe de seu gol ou algum jogador de seu time esteja mais próximo da bola ele fará imediatamente uma transição para o estado “Retornar à posição inicial”.

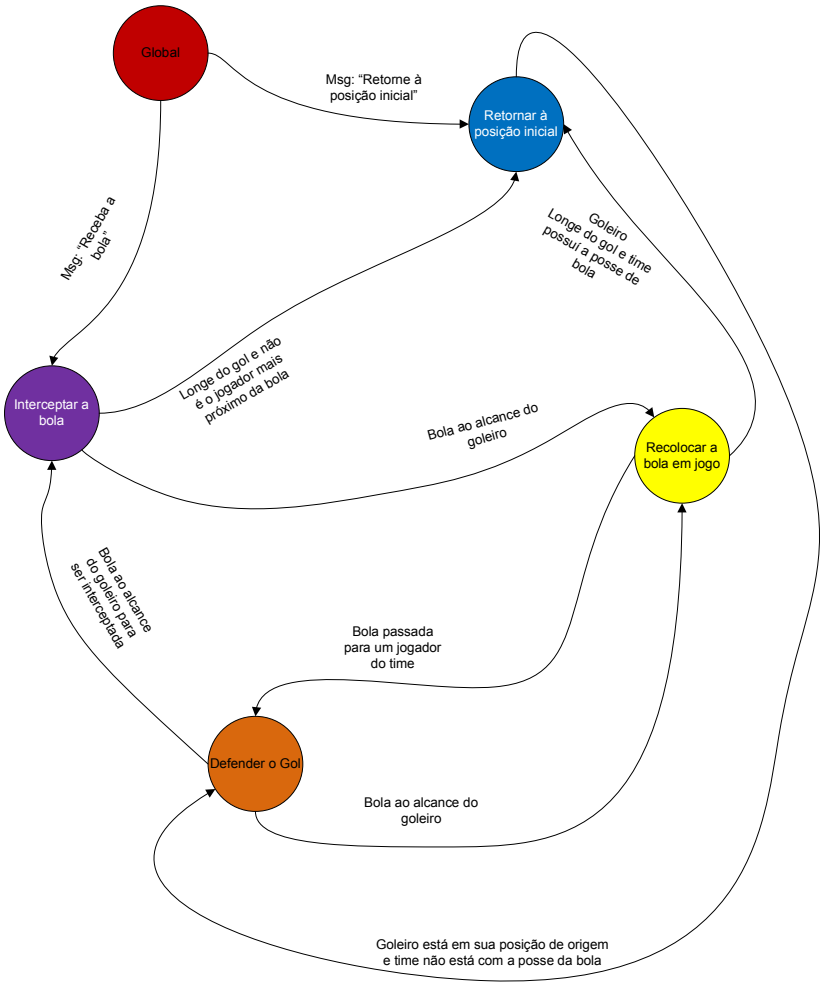

Figura 4.5 – Diagrama de estados da entidade Goleiro.

4.3.2 – Agentes Autônomos

Agentes autônomos são entidades programadas para interagir com o meio onde eles se situam.

Um agente autônomo deve ser capaz de analisar o ambiente e tomar ações baseadas no resultado de sua análise. Tais ações devem sempre visar o cumprimento de um determinado objetivo que pode ser fixo ou variar com o tempo.

Neste projeto a definição genérica dada para a autonomia de um agente computacional será discutida estritamente em termos de sua movimentação. Serão vistos todos os tipos de movimento considerados para um agente capaz de simular o comportamento de um jogador de futebol.

Podemos dividir a movimentação de um agente em três camadas:

1) Seleção da ação: Esta camada é responsável por decidir os objetivos a serem atingidos e qual deve ser o plano a ser seguido para se obter sucesso.

2) Direcionamento: Esta camada é responsável por calcular as trajetórias que devem ser seguidas. Para seguir uma trajetória, deve existir uma força resultante atuando sobre o agente que imprima uma determinada aceleração e seja capaz de movimentá-lo pelo caminho desejado.

3) Locomoção: Esta camada é responsável por determinar de que forma será feita a locomoção de "A" para "B". Agentes diferentes possuem diferentes formas de se movimentar através de uma mesma trajetória. Um cavalo, um coelho ou um jogador de futebol irão se locomover cada qual a sua maneira.

Os comportamentos considerados são explicados a seguir:

1) Chegada ao alvo: Neste comportamento o agente é submetido a uma força que o direciona para um alvo situado em uma determinada posição.

2) Afastamento: Este comportamento produz uma força que afasta o agente de um objeto situado em uma determinada posição. O comportamento é exatamente o oposto do que ocorre no item 1).

3) Chegada: A diferença para 1) é que neste comportamento ocorre uma desaceleração à medida que se aproxima do alvo. Por exemplo: Ao se aproximar de seu ponto de destino um automóvel irá desacelerar gradualmente até parar em posição final. Esta desaceleração produz uma ilusão de inteligência e um maior realismo.

4) Perseguição: Neste comportamento um agente se torna capaz de perseguir um alvo em movimento. A diferença para 1) é que a posição de "chegada" é constantemente recalculada tendo em vista a trajetória tomada pelo agente em fuga.

5) Evasão: Neste comportamento um agente se afasta de um perseguidor. Sua trajetória de fuga é feita utilizando-se o item 2). A diferença que o ponto de afastamento é constantemente recalculado tendo-se em vista a movimentação do perseguidor.

6) Caminhada Aleatória: Aqui um agente se movimenta de forma totalmente aleatória. Pode ser utilizado para simular caminhadas de animais ou pessoas, calcular probalilidades de determinados caminhos se cruzarem e etc. É criado um circulo de raio R centrado na posição do agente. Este deverá se movimentar para um ponto qualquer situado na circunferência.

7) Detecção de obstáculo: Aqui um agente detecta que existem pontos de intersecção entre seu perímetro e o perímetro de um determinado obstáculo. Dessa forma ele pode tomar ações como desviar, passar por cima ou até mesmo destruir dependendo da situação.

A cada instante todos os jogadores estão utilizando um ou mais dos comportamentos descritos acima. Dessa forma foi possível simular comportamentos tipicamente humanos em agentes autônomos programados.

Capítulo 5

Web Services

5.1 – Descrição

Neste capítulo definiremos o que são WS, suas aplicações, vantagens e a implementação deles neste projeto. Além disso, trataremos da linguagem utilizada para descrever os serviços dos WS (WSDL), do protocolo utilizado para codificar as chamadas às operações (SOAP) e da tecnologia empregada para facilitar o desenvolvimento de WS usando a linguagem Java (JAX-WS).

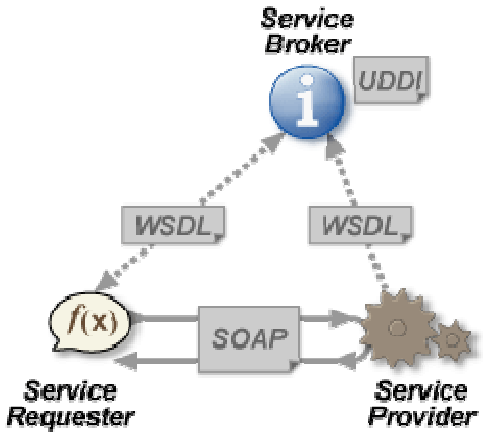

Figura 5.1 – Representação esquemática de um WS.

5.2 – Definição

Um WS nada mais é do que um programa que fornece algum serviço, com o objetivo de integrar sistemas ou ainda na comunicação entre aplicações diferentes. O uso de WS é fundamental em sistemas que trabalham em rede, uma vez que a integração por ele propiciada se dá mesmo entre programas que utilizem linguagens diferentes, ou ainda estejam em máquinas distintas.

O objetivo principal do WS é permitir a normalização da comunicação dentro de uma rede. Isto se dá da seguinte maneira:

1) O WS espera a requisição de um cliente, que especificará o serviço pretendido, além de enviar eventualmente parâmetros a serem considerados.

2) O WS executa as operações necessárias para processar a requisição do cliente.

3) O WS retorna ao cliente os dados pretendidos, utilizando o formato XML.

É o fato de trabalhar com um formato reconhecido e padronizado que faz com que WS sejam utilizados de forma satisfatória na comunicação entre aplicações distintas. Outra vantagem inerente a este tipo de aplicação é o isolamento do BD. Como os aplicativos-cliente não acessam diretamente os objetos que consultam o BD, há mais uma camada intermediária, o que facilita a segurança dos dados do sistema. Tudo sempre voltado à ideia de isolamento de camadas, seguindo o padrão MVC.

5.3 – SOAP

SOAP é um protocolo para invocar aplicações remotas, utilizando-se de uma linguagem padrão (o XML) e um mecanismo de transporte padrão (o HTTP).

O fato de SOAP ser amplamente utilizável, o que casa perfeitamente com WS, faz com que diversas linguagens de programação tenham bibliotecas específicas para a utilização deste protocolo. Ainda assim, o seu entendimento é vital para eventuais situações de erro ou problemas de interoperabilidade.

Uma mensagem SOAP é basicamente um documento XML, composto de três itens: envelope, header e body. Descreveremos brevemente, abaixo, como funciona uma mensagem SOAP.

Figura 5.2 – Mensagem SOAP

1) Envelope: é um elemento obrigatório da mensagem SOAP, sendo o elemento raiz do XML. Nele que é definido o conteúdo da mensagem, ou seja, nele é que se define que esta é uma mensagem SOAP. Opcionalmente, pode definir também o estilo de codificação.

2) Header (cabeçalho): é um elemento opcional. Caso exista, deve ser o primeiro elemento do envelope. Traz informações adicionais sobre a mensagem como, por exemplo, por que receptor (no caso de uma rede com muitos nós) a mensagem deve ser processada.

3) Body (corpo): é um elemento obrigatório. Nele é que estão contidas todas as informações que serão transportadas ao destino final. Opcionalmente, pode conter um elemento que carrega mensagens de status e erros retornadas pelos nós da rede que processam a mensagem.

5.4 – WSDL

A WSDL é uma linguagem baseada em XML e, portanto, compreendida pela maioria dos sistemas existentes, que tem o objetivo de descrever o WS. Para um determinado WS é, então, criado um documento WSDL, que define através de um XML Schema, entre outros, suas interfaces, operações e esquemas de codificação.

Abaixo seguem os passos do uso do documento WSDL em uma requisição a um WS:

1) O cliente obtém a descrição do WS que ele pretende utilizar, através do documento WSDL.

2) O cliente constrói a mensagem, baseando-se na descrição obtida anteriormente.

3) O cliente envia a mensagem ao WS.

4) O WS valida a mensagem, de acordo com as definições do documento WSDL.

Posteriormente é feito o processamento e o resultado é enviado ao cliente, mas isso já foge da alçada do WSDL.

5.5 – JAX–WS

JAX-WS é uma API do Java para a criação e o tratamento de WS. Ela fornece suporte a SOAP e tem como facilidade principal a simplificação do desenvolvimento dos WS e a redução no tempo de execução dos arquivos JAR.

O uso de JAX-WS neste projeto se dá, então, por razões óbvias. Ele atende aos requisitos de universalidade e de isolamento de camadas existentes em todo o projeto.

5.6 – Implementação

O WS foi construído para implementar toda a regra de negócio da aplicação de forma genérica.

Ele faz a intermediação entre qualquer cliente e o repositório de dados através de uma interface pública que expõe todas as funcionalidades do sistema através de entradas e saídas.

Dessa forma é possível separar completamente o desenvolvimento do cliente bem como do banco de dados, possibilitando desta forma a paralelização do desenvolvimento bem como uma maior facilidade na manutenção dos módulos, visto que não existe entrelaçamento entre camadas.

É possível desenvolver um cliente para qualquer plataforma, contanto que este implemente a comunicação prevista na interface pública do WS.

Esta interface pública prevê as seguintes funcionalidades:

Entradas:

Registro de usuário

Compra de créditos

Criação de um clube

Criação de um campeonato

Login de usuário

Logout de usuário

Incrementar atributos de um jogador

Registrar clube em um campeonato

Saídas:

Informações sobre um usuário

Informações sobre um clube

Informações sobre um campeonato

Informações sobre os jogos de uma rodada

Informações sobre campeonatos iniciados e não-iniciados

Classe que define a interface pública do Web Service:

/.../

```
@WebService
public interface FSZWebService {

   @WebMethod
   public WSOperationContext doRegisterUser(String loginName, String password);
   @WebMethod
   public WSOperationContext doLogin(String Login, String password);
   @WebMethod
   public WSOperationContext doLogout(String username, String password);
   @WebMethod
   public UserOperationContext getUserContext(String requesterName, String requestPwd, String
username);
   @WebMethod
   public WSOperationContext doBuyCredits(String username, String password, int creditsToBuy);
   @WebMethod
   public WSOperationContext doCreateClub(String username, String password, String clubName);
   @WebMethod
   public ClubOperationContext getClubContext(String requesterName, String password, String
clubName);
```

```
  @WebMethod
  public WSOperationContext doCreateLeague(String username, String password, String leagueName,
int targetClubs);
  @WebMethod
  public LeagueOperationContext getLeagueContext(String requesterName, String password, String
leagueName);
  @WebMethod
  public LeagueInfoOperationContext getLeagueInfo(String requesterName, String password, String
leagueName);
  @WebMethod
  public LeagueRoundOperationContext getLeagueRound(String requesterName, String password,
String leagueName, int roundNumber);
  @WebMethod
  public LeagueInfoListOperationContext getLeaguesByOwner(String requesterName, String password,
String leaguesOwnerName);
  @WebMethod
  public LeagueInfoListOperationContext getLeaguesStarted(String requesterName, String password);
  @WebMethod
  public LeagueInfoListOperationContext getLeaguesNotStarted(String requesterName, String
password);
  @WebMethod
  public WSOperationContext doIncrementPlayerAttribute(String login, String password, String
clubName,
                                        int playerNumber, String playerAttribute,
                                        int attrIncrement);
  @WebMethod
  public WSOperationContext doRegisterClubInLeague(String login, String password, String clubName,
String leagueName);
}
```

A interface pública descrita acima é traduzida para o padrão WSDL.

A tradução entre a classe acima e o WSDL é feita pelo utilitário "wsgen".

Este utilitário recebe como entrada uma classe que define a interface do Web Service (como a classe acima) e produz como saída um documento WSDL.

Este documento é publicado no IP e porta especificado. Através deste documento é possível implementar um cliente compatível com o Web Service.

O utilitário wsimport é capaz de ler o documento WSDL é gerar todo o código java de infraestrutura necessário para o cliente se comunicar com o Web

Service.

Todos os métodos descritos acima estão contidos no WSDL:

WSDL da aplicação

```
<!--
 Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.6 in JDK 6.
--

<!--
Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.6 in JDK 6.
-->

<definitions targetNamespace="http://fszservice/" name="FSZWebServiceImplService">

   <types>

      <xsd:schema>
         <xsd:import namespace="http://fszservice/" schemaLocation="http://localhost:9999/fsz?xsd=1"/>
      </xsd:schema>
   </types>

   <message name="doRegisterUser">
      <part name="parameters" element="tns:doRegisterUser"/>
   </message>

   <message name="doRegisterUserResponse">
      <part name="parameters" element="tns:doRegisterUserResponse"/>
   </message>

   <message name="doLogin">
      <part name="parameters" element="tns:doLogin"/>
   </message>

   <message name="doLoginResponse">
      <part name="parameters" element="tns:doLoginResponse"/>
   </message>
```

```
<message name="doLogout">
   <part name="parameters" element="tns:doLogout"/>
</message>

<message name="doLogoutResponse">
   <part name="parameters" element="tns:doLogoutResponse"/>
</message>

<message name="getUserContext">
   <part name="parameters" element="tns:getUserContext"/>
</message>

<message name="getUserContextResponse">
   <part name="parameters" element="tns:getUserContextResponse"/>
</message>

<message name="doBuyCredits">
   <part name="parameters" element="tns:doBuyCredits"/>
</message>

<message name="doBuyCreditsResponse">
   <part name="parameters" element="tns:doBuyCreditsResponse"/>
</message>

<message name="doCreateClub">
   <part name="parameters" element="tns:doCreateClub"/>
</message>

<message name="doCreateClubResponse">
   <part name="parameters" element="tns:doCreateClubResponse"/>
</message>

<message name="getClubContext">
   <part name="parameters" element="tns:getClubContext"/>
</message>

<message name="getClubContextResponse">
   <part name="parameters" element="tns:getClubContextResponse"/>
</message>
```

```
<message name="doCreateLeague">
   <part name="parameters" element="tns:doCreateLeague"/>
</message>

<message name="doCreateLeagueResponse">
   <part name="parameters" element="tns:doCreateLeagueResponse"/>
</message>

<message name="getLeagueContext">
   <part name="parameters" element="tns:getLeagueContext"/>
</message>

<message name="getLeagueContextResponse">
   <part name="parameters" element="tns:getLeagueContextResponse"/>
</message>

<message name="getLeagueInfo">
   <part name="parameters" element="tns:getLeagueInfo"/>
</message>

<message name="getLeagueInfoResponse">
   <part name="parameters" element="tns:getLeagueInfoResponse"/>
</message>

<message name="getLeagueRound">
   <part name="parameters" element="tns:getLeagueRound"/>
</message>

<message name="getLeagueRoundResponse">
   <part name="parameters" element="tns:getLeagueRoundResponse"/>
</message>

<message name="getLeaguesByOwner">
   <part name="parameters" element="tns:getLeaguesByOwner"/>
</message>

<message name="getLeaguesByOwnerResponse">
   <part name="parameters" element="tns:getLeaguesByOwnerResponse"/>
</message>
```

```
<message name="getLeaguesStarted">
   <part name="parameters" element="tns:getLeaguesStarted"/>
</message>

<message name="getLeaguesStartedResponse">
   <part name="parameters" element="tns:getLeaguesStartedResponse"/>
</message>

<message name="getLeaguesNotStarted">
   <part name="parameters" element="tns:getLeaguesNotStarted"/>
</message>

<message name="getLeaguesNotStartedResponse">
   <part name="parameters" element="tns:getLeaguesNotStartedResponse"/>
</message>

<message name="doIncrementPlayerAttribute">
   <part name="parameters" element="tns:doIncrementPlayerAttribute"/>
</message>

<message name="doIncrementPlayerAttributeResponse">
   <part name="parameters" element="tns:doIncrementPlayerAttributeResponse"/>
</message>

<message name="doRegisterClubInLeague">
   <part name="parameters" element="tns:doRegisterClubInLeague"/>
</message>

<message name="doRegisterClubInLeagueResponse">
   <part name="parameters" element="tns:doRegisterClubInLeagueResponse"/>
</message>

<portType name="FSZWebService">

   <operation name="doRegisterUser">
      <input message="tns:doRegisterUser"/>
      <output message="tns:doRegisterUserResponse"/>
   </operation>

   <operation name="doLogin">
```

```
    <input message="tns:doLogin"/>
    <output message="tns:doLoginResponse"/>
</operation>

<operation name="doLogout">
    <input message="tns:doLogout"/>
    <output message="tns:doLogoutResponse"/>
</operation>

<operation name="getUserContext">
    <input message="tns:getUserContext"/>
    <output message="tns:getUserContextResponse"/>
</operation>

<operation name="doBuyCredits">
    <input message="tns:doBuyCredits"/>
    <output message="tns:doBuyCreditsResponse"/>
</operation>

<operation name="doCreateClub">
    <input message="tns:doCreateClub"/>
    <output message="tns:doCreateClubResponse"/>
</operation>

<operation name="getClubContext">
    <input message="tns:getClubContext"/>
    <output message="tns:getClubContextResponse"/>
</operation>

<operation name="doCreateLeague">
    <input message="tns:doCreateLeague"/>
    <output message="tns:doCreateLeagueResponse"/>
</operation>

<operation name="getLeagueContext">
    <input message="tns:getLeagueContext"/>
    <output message="tns:getLeagueContextResponse"/>
</operation>

<operation name="getLeagueInfo">
```

```
      <input message="tns:getLeagueInfo"/>
      <output message="tns:getLeagueInfoResponse"/>
    </operation>

    <operation name="getLeagueRound">
      <input message="tns:getLeagueRound"/>
      <output message="tns:getLeagueRoundResponse"/>
    </operation>

    <operation name="getLeaguesByOwner">
      <input message="tns:getLeaguesByOwner"/>
      <output message="tns:getLeaguesByOwnerResponse"/>
    </operation>

    <operation name="getLeaguesStarted">
      <input message="tns:getLeaguesStarted"/>
      <output message="tns:getLeaguesStartedResponse"/>
    </operation>

    <operation name="getLeaguesNotStarted">
      <input message="tns:getLeaguesNotStarted"/>
      <output message="tns:getLeaguesNotStartedResponse"/>
    </operation>

    <operation name="doIncrementPlayerAttribute">
      <input message="tns:doIncrementPlayerAttribute"/>
      <output message="tns:doIncrementPlayerAttributeResponse"/>
    </operation>

    <operation name="doRegisterClubInLeague">
      <input message="tns:doRegisterClubInLeague"/>
      <output message="tns:doRegisterClubInLeagueResponse"/>
    </operation>
  </portType>

  <binding name="FSZWebServiceImplPortBinding" type="tns:FSZWebService">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>

    <operation name="doRegisterUser">
      <soap:operation soapAction=""/>
```

```
    <input>
      <soap:body use="literal"/>
    </input>

    <output>
      <soap:body use="literal"/>
    </output>
  </operation>

  <operation name="doLogin">
    <soap:operation soapAction=""/>

    <input>
      <soap:body use="literal"/>
    </input>

    <output>
      <soap:body use="literal"/>
    </output>
  </operation>

  <operation name="doLogout">
    <soap:operation soapAction=""/>

    <input>
      <soap:body use="literal"/>
    </input>

    <output>
      <soap:body use="literal"/>
    </output>
  </operation>

  <operation name="getUserContext">
    <soap:operation soapAction=""/>

    <input>
      <soap:body use="literal"/>
    </input>
```

```
    <output>
      <soap:body use="literal"/>
    </output>
</operation>

<operation name="doBuyCredits">
    <soap:operation soapAction=""/>

    <input>
      <soap:body use="literal"/>
    </input>

    <output>
      <soap:body use="literal"/>
    </output>
</operation>

<operation name="doCreateClub">
    <soap:operation soapAction=""/>

    <input>
      <soap:body use="literal"/>
    </input>

    <output>
      <soap:body use="literal"/>
    </output>
</operation>

<operation name="getClubContext">
    <soap:operation soapAction=""/>

    <input>
      <soap:body use="literal"/>
    </input>

    <output>
      <soap:body use="literal"/>
    </output>
```

```
</operation>

<operation name="doCreateLeague">
   <soap:operation soapAction=""/>

   <input>
      <soap:body use="literal"/>
   </input>

   <output>
      <soap:body use="literal"/>
   </output>
</operation>

<operation name="getLeagueContext">
   <soap:operation soapAction=""/>

   <input>
      <soap:body use="literal"/>
   </input>

   <output>
      <soap:body use="literal"/>
   </output>
</operation>

<operation name="getLeagueInfo">
   <soap:operation soapAction=""/>

   <input>
      <soap:body use="literal"/>
   </input>

   <output>
      <soap:body use="literal"/>
   </output>
</operation>

<operation name="getLeagueRound">
   <soap:operation soapAction=""/>
```

```
    <input>
      <soap:body use="literal"/>
    </input>

    <output>
      <soap:body use="literal"/>
    </output>
</operation>

<operation name="getLeaguesByOwner">
    <soap:operation soapAction=""/>

    <input>
      <soap:body use="literal"/>
    </input>

    <output>
      <soap:body use="literal"/>
    </output>
</operation>

<operation name="getLeaguesStarted">
    <soap:operation soapAction=""/>

    <input>
      <soap:body use="literal"/>
    </input>

    <output>
      <soap:body use="literal"/>
    </output>
</operation>

<operation name="getLeaguesNotStarted">
    <soap:operation soapAction=""/>

    <input>
      <soap:body use="literal"/>
    </input>
```

```
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>

        <operation name="doIncrementPlayerAttribute">
            <soap:operation soapAction=""/>

            <input>
                <soap:body use="literal"/>
            </input>

            <output>
                <soap:body use="literal"/>
            </output>
        </operation>

        <operation name="doRegisterClubInLeague">
            <soap:operation soapAction=""/>

            <input>
                <soap:body use="literal"/>
            </input>

            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>

    <service name="FSZWebServiceImplService">

        <port name="FSZWebServiceImplPort" binding="tns:FSZWebServiceImplPortBinding">
            <soap:address location="http://localhost:9999/fsz"/>
        </port>
    </service>
</definitions>
```

Definição dos tipos utilizados pelo WSDL:

```
<!--
 Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.6 in JDK 6.
-->

<xs:schema version="1.0" targetNamespace="http://fszservice/">
   <xs:element name="doBuyCredits" type="tns:doBuyCredits"/>
   <xs:element name="doBuyCreditsResponse" type="tns:doBuyCreditsResponse"/>
   <xs:element name="doCreateClub" type="tns:doCreateClub"/>
   <xs:element name="doCreateClubResponse" type="tns:doCreateClubResponse"/>
   <xs:element name="doCreateLeague" type="tns:doCreateLeague"/>
   <xs:element name="doCreateLeagueResponse" type="tns:doCreateLeagueResponse"/>
   <xs:element name="doIncrementPlayerAttribute" type="tns:doIncrementPlayerAttribute"/>
   <xs:element                                              name="doIncrementPlayerAttributeResponse"
type="tns:doIncrementPlayerAttributeResponse"/>
   <xs:element name="doLogin" type="tns:doLogin"/>
   <xs:element name="doLoginResponse" type="tns:doLoginResponse"/>
   <xs:element name="doLogout" type="tns:doLogout"/>
   <xs:element name="doLogoutResponse" type="tns:doLogoutResponse"/>
   <xs:element name="doRegisterClubInLeague" type="tns:doRegisterClubInLeague"/>
   <xs:element                                                name="doRegisterClubInLeagueResponse"
type="tns:doRegisterClubInLeagueResponse"/>
   <xs:element name="doRegisterUser" type="tns:doRegisterUser"/>
   <xs:element name="doRegisterUserResponse" type="tns:doRegisterUserResponse"/>
   <xs:element name="getClubContext" type="tns:getClubContext"/>
   <xs:element name="getClubContextResponse" type="tns:getClubContextResponse"/>
   <xs:element name="getLeagueContext" type="tns:getLeagueContext"/>
   <xs:element name="getLeagueContextResponse" type="tns:getLeagueContextResponse"/>
   <xs:element name="getLeagueInfo" type="tns:getLeagueInfo"/>
   <xs:element name="getLeagueInfoResponse" type="tns:getLeagueInfoResponse"/>
   <xs:element name="getLeagueRound" type="tns:getLeagueRound"/>
   <xs:element name="getLeagueRoundResponse" type="tns:getLeagueRoundResponse"/>
   <xs:element name="getLeaguesByOwner" type="tns:getLeaguesByOwner"/>
   <xs:element name="getLeaguesByOwnerResponse" type="tns:getLeaguesByOwnerResponse"/>
   <xs:element name="getLeaguesNotStarted" type="tns:getLeaguesNotStarted"/>
   <xs:element name="getLeaguesNotStartedResponse" type="tns:getLeaguesNotStartedResponse"/>
   <xs:element name="getLeaguesStarted" type="tns:getLeaguesStarted"/>
   <xs:element name="getLeaguesStartedResponse" type="tns:getLeaguesStartedResponse"/>
```

```
    <xs:element name="getUserContext" type="tns:getUserContext"/>
    <xs:element name="getUserContextResponse" type="tns:getUserContextResponse"/>

    <xs:complexType name="doCreateLeague">

      <xs:sequence>
        <xs:element name="arg0" type="xs:string" minOccurs="0"/>
        <xs:element name="arg1" type="xs:string" minOccurs="0"/>
        <xs:element name="arg2" type="xs:string" minOccurs="0"/>
        <xs:element name="arg3" type="xs:int"/>
      </xs:sequence>
    </xs:complexType>

    <xs:complexType name="doCreateLeagueResponse">

      <xs:sequence>
        <xs:element name="return" type="tns:wsOperationContext" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>

    <xs:complexType name="wsOperationContext">

      <xs:sequence>
        <xs:element name="OK" type="xs:boolean"/>
        <xs:element name="statusCode" type="tns:operationStatus" minOccurs="0"/>
        <xs:element name="statusCodeParameters" type="xs:string" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>

    <xs:complexType name="doCreateClub">

      <xs:sequence>
        <xs:element name="arg0" type="xs:string" minOccurs="0"/>
        <xs:element name="arg1" type="xs:string" minOccurs="0"/>
        <xs:element name="arg2" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>

    <xs:complexType name="doCreateClubResponse">
```

```
    <xs:sequence>
      <xs:element name="return" type="tns:wsOperationContext" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="getClubContext">

    <xs:sequence>
      <xs:element name="arg0" type="xs:string" minOccurs="0"/>
      <xs:element name="arg1" type="xs:string" minOccurs="0"/>
      <xs:element name="arg2" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="getClubContextResponse">

    <xs:sequence>
      <xs:element name="return" type="tns:clubOperationContext" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="clubOperationContext">

    <xs:complexContent>

      <xs:extension base="tns:wsOperationContext">

        <xs:sequence>
          <xs:element name="clubContext" type="tns:clubContext" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="clubContext">

    <xs:complexContent>

      <xs:extension base="tns:wsBean">
```

```
            <xs:sequence>
               <xs:element name="leagueName" type="xs:string" minOccurs="0"/>
               <xs:element name="name" type="xs:string" minOccurs="0"/>
               <xs:element name="owner" type="xs:string" minOccurs="0"/>
               <xs:element name="players" type="tns:playerContext" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
               <xs:element name="registeredInLeague" type="xs:boolean"/>
               <xs:element name="standingsContext" type="tns:clubStandingsContext" minOccurs="0"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

   <xs:complexType name="wsBean">

      <xs:sequence>
         <xs:element name="type" type="tns:wsBeanType" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="playerContext">

      <xs:complexContent>

         <xs:extension base="tns:wsBean">

            <xs:sequence>
               <xs:element name="clubName" type="xs:string" minOccurs="0"/>
               <xs:element name="dribble" type="xs:int"/>
               <xs:element name="kick" type="xs:int"/>
               <xs:element name="name" type="xs:string" minOccurs="0"/>
               <xs:element name="number" type="xs:int"/>
               <xs:element name="pass" type="xs:int"/>
               <xs:element name="speed" type="xs:int"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
```

```
<xs:complexType name="clubStandingsContext">

   <xs:complexContent>

      <xs:extension base="tns:wsBean">

         <xs:sequence>
            <xs:element name="clubName" type="xs:string" minOccurs="0"/>
            <xs:element name="draws" type="xs:int"/>
            <xs:element name="gamesPlayed" type="xs:int"/>
            <xs:element name="goalsAgainst" type="xs:int"/>
            <xs:element name="goalsDifference" type="xs:int"/>
            <xs:element name="goalsFor" type="xs:int"/>
            <xs:element name="leagueName" type="xs:string" minOccurs="0"/>
            <xs:element name="looses" type="xs:int"/>
            <xs:element name="points" type="xs:int"/>
            <xs:element name="pointsPercentage" type="xs:int"/>
            <xs:element name="position" type="xs:int"/>
            <xs:element name="victories" type="xs:int"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>

<xs:complexType name="doRegisterClubInLeague">

   <xs:sequence>
      <xs:element name="arg0" type="xs:string" minOccurs="0"/>
      <xs:element name="arg1" type="xs:string" minOccurs="0"/>
      <xs:element name="arg2" type="xs:string" minOccurs="0"/>
      <xs:element name="arg3" type="xs:string" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="doRegisterClubInLeagueResponse">

   <xs:sequence>
      <xs:element name="return" type="tns:wsOperationContext" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>
```

```
  <xs:complexType name="getLeaguesNotStarted">

    <xs:sequence>
      <xs:element name="arg0" type="xs:string" minOccurs="0"/>
      <xs:element name="arg1" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="getLeaguesNotStartedResponse">

    <xs:sequence>
      <xs:element name="return" type="tns:leagueInfoListOperationContext" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="leagueInfoListOperationContext">

    <xs:complexContent>

      <xs:extension base="tns:wsOperationContext">

        <xs:sequence>
          <xs:element name="infoListContext" type="tns:leagueInfoListContext" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="leagueInfoListContext">

    <xs:complexContent>

      <xs:extension base="tns:wsBean">

        <xs:sequence>
          <xs:element name="infoList" type="tns:leagueInfoContext" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
```

```
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="leagueInfoContext">

    <xs:complexContent>

      <xs:extension base="tns:wsBean">

        <xs:sequence>
          <xs:element name="clubStandings" type="tns:clubStandingsContext" nillable="true"
minOccurs="0" maxOccurs="unbounded"/>
          <xs:element name="finished" type="xs:boolean"/>
          <xs:element name="hours" type="xs:int"/>
          <xs:element name="minutes" type="xs:int"/>
          <xs:element name="name" type="xs:string" minOccurs="0"/>
          <xs:element name="nextRound" type="xs:int"/>
          <xs:element name="owner" type="xs:string" minOccurs="0"/>
          <xs:element name="seconds" type="xs:int"/>
          <xs:element name="started" type="xs:boolean"/>
          <xs:element name="targetClubsCount" type="xs:int"/>
          <xs:element name="totalRounds" type="xs:int"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="getLeaguesStarted">

    <xs:sequence>
      <xs:element name="arg0" type="xs:string" minOccurs="0"/>
      <xs:element name="arg1" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="getLeaguesStartedResponse">

    <xs:sequence>
      <xs:element name="return" type="tns:leagueInfoListOperationContext" minOccurs="0"/>
    </xs:sequence>
```

```
</xs:complexType>

<xs:complexType name="doBuyCredits">

   <xs:sequence>
      <xs:element name="arg0" type="xs:string" minOccurs="0"/>
      <xs:element name="arg1" type="xs:string" minOccurs="0"/>
      <xs:element name="arg2" type="xs:int"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="doBuyCreditsResponse">

   <xs:sequence>
      <xs:element name="return" type="tns:wsOperationContext" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="getLeagueContext">

   <xs:sequence>
      <xs:element name="arg0" type="xs:string" minOccurs="0"/>
      <xs:element name="arg1" type="xs:string" minOccurs="0"/>
      <xs:element name="arg2" type="xs:string" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="getLeagueContextResponse">

   <xs:sequence>
      <xs:element name="return" type="tns:leagueOperationContext" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="leagueOperationContext">

   <xs:complexContent>

      <xs:extension base="tns:wsOperationContext">
```

```
            <xs:sequence>
              <xs:element name="leagueContext" type="tns:leagueContext" minOccurs="0"/>
            </xs:sequence>
          </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="leagueContext">

        <xs:complexContent>

          <xs:extension base="tns:wsBean">

            <xs:sequence>
              <xs:element name="fixtures" type="tns:roundContext" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
              <xs:element name="leagueInfoContext" type="tns:leagueInfoContext" minOccurs="0"/>
            </xs:sequence>
          </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="roundContext">

        <xs:complexContent>

          <xs:extension base="tns:wsBean">

            <xs:sequence>
              <xs:element name="matches" type="tns:roundMatchContext" nillable="true"
minOccurs="0" maxOccurs="unbounded"/>
              <xs:element name="number" type="xs:int"/>
            </xs:sequence>
          </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="roundMatchContext">

        <xs:complexContent>
```

```
      <xs:extension base="tns:wsBean">

        <xs:sequence>
          <xs:element name="duration" type="xs:int"/>
          <xs:element name="goalsTeam1" type="xs:int"/>
          <xs:element name="goalsTeam2" type="xs:int"/>
          <xs:element name="leagueName" type="xs:string" minOccurs="0"/>
          <xs:element name="matchPlayed" type="xs:boolean"/>
          <xs:element name="roundNumber" type="xs:int"/>
          <xs:element name="seed" type="xs:int"/>
          <xs:element name="team1" type="xs:string" minOccurs="0"/>
          <xs:element name="team2" type="xs:string" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="doRegisterUser">

    <xs:sequence>
      <xs:element name="arg0" type="xs:string" minOccurs="0"/>
      <xs:element name="arg1" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="doRegisterUserResponse">

    <xs:sequence>
      <xs:element name="return" type="tns:wsOperationContext" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="getLeaguesByOwner">

    <xs:sequence>
      <xs:element name="arg0" type="xs:string" minOccurs="0"/>
      <xs:element name="arg1" type="xs:string" minOccurs="0"/>
      <xs:element name="arg2" type="xs:string" minOccurs="0"/>
    </xs:sequence>
```

```
</xs:complexType>

<xs:complexType name="getLeaguesByOwnerResponse">

   <xs:sequence>
      <xs:element name="return" type="tns:leagueInfoListOperationContext" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="doIncrementPlayerAttribute">

   <xs:sequence>
      <xs:element name="arg0" type="xs:string" minOccurs="0"/>
      <xs:element name="arg1" type="xs:string" minOccurs="0"/>
      <xs:element name="arg2" type="xs:string" minOccurs="0"/>
      <xs:element name="arg3" type="xs:int"/>
      <xs:element name="arg4" type="xs:string" minOccurs="0"/>
      <xs:element name="arg5" type="xs:int"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="doIncrementPlayerAttributeResponse">

   <xs:sequence>
      <xs:element name="return" type="tns:wsOperationContext" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="doLogin">

   <xs:sequence>
      <xs:element name="arg0" type="xs:string" minOccurs="0"/>
      <xs:element name="arg1" type="xs:string" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="doLoginResponse">

   <xs:sequence>
      <xs:element name="return" type="tns:wsOperationContext" minOccurs="0"/>
```

```
    </xs:sequence>
</xs:complexType>

<xs:complexType name="getUserContext">

    <xs:sequence>
        <xs:element name="arg0" type="xs:string" minOccurs="0"/>
        <xs:element name="arg1" type="xs:string" minOccurs="0"/>
        <xs:element name="arg2" type="xs:string" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="getUserContextResponse">

    <xs:sequence>
        <xs:element name="return" type="tns:userOperationContext" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="userOperationContext">

    <xs:complexContent>

        <xs:extension base="tns:wsOperationContext">

            <xs:sequence>
                <xs:element name="userContext" type="tns:userContext" minOccurs="0"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="userContext">

    <xs:complexContent>

        <xs:extension base="tns:wsBean">

            <xs:sequence>
```

```
            <xs:element name="clubs" type="xs:string" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element name="credits" type="xs:int"/>
            <xs:element name="leagues" type="xs:string" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element name="logged" type="xs:boolean"/>
            <xs:element name="maxCredits" type="xs:int"/>
            <xs:element name="name" type="xs:string" minOccurs="0"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="getLeagueRound">

      <xs:sequence>
        <xs:element name="arg0" type="xs:string" minOccurs="0"/>
        <xs:element name="arg1" type="xs:string" minOccurs="0"/>
        <xs:element name="arg2" type="xs:string" minOccurs="0"/>
        <xs:element name="arg3" type="xs:int"/>
      </xs:sequence>
    </xs:complexType>

    <xs:complexType name="getLeagueRoundResponse">

      <xs:sequence>
        <xs:element name="return" type="tns:leagueRoundOperationContext" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>

    <xs:complexType name="leagueRoundOperationContext">

      <xs:complexContent>

        <xs:extension base="tns:wsOperationContext">

          <xs:sequence>
            <xs:element name="roundContext" type="tns:roundContext" minOccurs="0"/>
          </xs:sequence>
        </xs:extension>
```

```
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="doLogout">

    <xs:sequence>
        <xs:element name="arg0" type="xs:string" minOccurs="0"/>
        <xs:element name="arg1" type="xs:string" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="doLogoutResponse">

    <xs:sequence>
        <xs:element name="return" type="tns:wsOperationContext" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="getLeagueInfo">

    <xs:sequence>
        <xs:element name="arg0" type="xs:string" minOccurs="0"/>
        <xs:element name="arg1" type="xs:string" minOccurs="0"/>
        <xs:element name="arg2" type="xs:string" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="getLeagueInfoResponse">

    <xs:sequence>
        <xs:element name="return" type="tns:leagueInfoOperationContext" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="leagueInfoOperationContext">

    <xs:complexContent>

        <xs:extension base="tns:wsOperationContext">
```

```
            <xs:sequence>
              <xs:element name="leagueInfoContext" type="tns:leagueInfoContext" minOccurs="0"/>
            </xs:sequence>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>

      <xs:simpleType name="operationStatus">

        <xs:restriction base="xs:string">
          <xs:enumeration value="WS_STATUS_OK"/>
          <xs:enumeration value="WS_STATUS_AUTH_ERROR"/>
          <xs:enumeration value="WS_STATUS_USER_ALREADY_REGISTERED"/>
          <xs:enumeration value="WS_STATUS_USER_NOT_REGISTERED"/>
          <xs:enumeration value="WS_STATUS_USER_NOT_LOGGED"/>
          <xs:enumeration value="WS_STATUS_INSUFFICIENT_CREDITS"/>
          <xs:enumeration value="WS_STATUS_INVALID_PLAYER_ATTRIBUTE_VALUE"/>
          <xs:enumeration value="WS_STATUS_LEAGUE_NOT_REGISTERED"/>
          <xs:enumeration value="WS_STATUS_LEAGUE_ALREADY_EXIST"/>
          <xs:enumeration value="WS_STATUS_INVALID_LEAGUE_ROUND"/>
          <xs:enumeration value="WS_STATUS_INVALID_MATCH"/>
          <xs:enumeration value="WS_STATUS_INVALID_CREDITS_VALUE"/>
          <xs:enumeration value="WS_STATUS_CLUB_ALREADY_EXIST"/>
          <xs:enumeration value="WS_STATUS_CLUB_NOT_REGISTERED"/>
          <xs:enumeration value="WS_STATUS_PLAYER_NOT_FOUND"/>
          <xs:enumeration value="WS_STATUS_INVALID_PLAYER_ATTRIBUTE"/>
          <xs:enumeration value="WS_STATUS_LEAGUE_ALREADY_STARTED"/>
          <xs:enumeration value="WS_STATUS_CLUB_ALREADY_REGISTERED_IN_LEAGUE"/>
          <xs:enumeration
value="WS_STATUS_USER_ALREADY_HAS_CLUB_IN_THIS_LEAGUE"/>
          <xs:enumeration value="WS_STATUS_LEAGUE_NOT_STARTED"/>
          <xs:enumeration value="WS_STATUS_LEAGUE_ALREADY_FINISHED"/>
        </xs:restriction>
      </xs:simpleType>

      <xs:simpleType name="wsBeanType">

        <xs:restriction base="xs:string">
          <xs:enumeration value="WS_BEAN_CLUB"/>
          <xs:enumeration value="WS_BEAN_CLUB_STANDINGS"/>
```

```
            <xs:enumeration value="WS_BEAN_LEAGUE"/>
            <xs:enumeration value="WS_BEAN_LEAGUE_INFO"/>
            <xs:enumeration value="WS_BEAN_LEAGUE_INFO_LIST"/>
            <xs:enumeration value="WS_BEAN_LEAGUE_STANDINGS"/>
            <xs:enumeration value="WS_BEAN_PLAYER"/>
            <xs:enumeration value="WS_BEAN_ROUND"/>
            <xs:enumeration value="WS_BEAN_ROUND_MATCH"/>
            <xs:enumeration value="WS_BEAN_USER"/>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

Capítulo 6

Conclusão

Neste capítulo trataremos basicamente dos resultados obtidos com a realização deste projeto. Abordaremos, sobretudo, a complexidade observada ao longo do desenvolvimento do sistema e a validade das escolhas das tecnologias utilizadas.

Desde o começo era sabido que o projeto apresentaria um nível elevado de complexidade, tamanha era a quantidade de diferentes tecnologias a serem estudadas, compreendidas e empregadas. Mesmo que estas tecnologias visassem a garantir a manutebilidade, o isolamento de camadas segundo o padrão MVC e a ideia de um sistema multiplataforma, sua aplicação teve de ser cirteriosa a fim de se evitar pequenos problemas que, ao fim do projeto, significassem grandes problemas.

Para que se pudesse evitar este tipo de situação, foi seguido à risca o ideal de desenvolvimento em vários ciclos, nos quais se incluíam testes por módulo. Além destes, houve ainda testes de inegração entre os módulos, de vital importância para que se reduzisse a complexidade do projeto, pelo menos no âmbito de testes.

De todos os módulos desenvolvidos aquele que, sem dúvidas, configurou-se o mais complexo foi o de inteligência artificial. Pode-se afirmar que isto já era esperado, uma vez que este módulo é o centro de todo o sistema. A construção da MEF nem se apresenta como algo de elevada dificuldade, uma vez que ela representa apenas uma abstração a ser considerada para a configuração dos agentes inteligentes. No entanto, passar para código-fonte todas as interações possíveis de uma MEF de duas camadas é algo complexo o suficiente para demandar tempo de pesquisa, codificação e testes.

Outro módulo que apresentou severas dificuldades foi o módulo de BD. Neste caso, especificamente, a dificuldade se apresentou na forma da organização de operações complexas que envolvessem várias classes de DAOs. Assegurar que o acesso ao banco fosse feito de forma simultânea, gerar todas as alterações necessárias a cada operação complexa e assegurar a integridade dos dados que circulam através de classes de tabelas demanda uma série de testes criteriosos.

Estes testes tratam, basicamente, de verificar as condições extremas no acesso ao banco, tentando a inclusão de dados fora do escopo especificado ou no limite dele, por exemplo. Além disso, a verificação manual de todos os dados alterados teve de ser feita,

para que se assegurasse a persistência adequada dos dados gerados. Este trabalho, basicamente mecânico, demanda atenção e tempo.

Os demais módulos, ao passo que apresentavam dificuldades próprias inerentes a eles, foram facilitados por uma série de ferramentas que possibilitavam seus desenvolvimentos de forma rápida e segura. De maneira geral, os módulos que menos ferramentas específicas utilizaram foram justamente os de IA e de BD.

Os resultados obtidos, ao final do projeto, foram bastante satisfatórios. O funcionamento do jogo se deu da maneira esperada mas, mais do que apenas isso, as tecnologias empregadas para produzi-lo se mostraram eficientes e adequadas. Todas elas permitiram que o desenvolvimento do projeto fosse o melhor possível, assegurando o cumprimento de seus requisitos e garantindo sua aplicabilidade prática pretendida.

O principal destaque neste quesito de tecnologias empregadas vai para o Facelets. Por ser uma tecnologia ainda não largamente empregada, e a despeito de toda a documentação e relatos colhidos das mais diversas fontes, o desempenho desta tecnologia mostrava-se a maior das incógnitas de desenvolvimento do projeto.

Seu funcionamento mostrou-se adequado à especificação pretendida, garantindo, entre outros, a manutebilidade, o isolamento de camadas e o desenvolvimento rápido da parte de visão da interface WEB.

Por experiências anteriores com a tecnologia JSP, pode-se dizer que o uso de Facelets apresenta considerável avanço em relação àquela. Além de reutilização de componentes e de uma melhor reportagem de erros, o desempenho do sistema torna-se melhor, aumentando sua velocidade. Além disso, o uso de Facelets isola de maneira total o código da visão, permitindo uma manutebilidade muito mais eficaz e segura, se a relacionarmos com a que era obtida através do uso de páginas JSP.

Podemos dizer, então, que o projeto mostrou-se um sucesso, atingindo todos os seus objetivos e permitindo, caso seja de interesse de desenvolvedores, ampliá-lo e melhorá-lo com acréscimos de artefatos, reutilizando-se os já existentes.

Bibliografia

[1] GOODMAN, D., *JavaScript & DHTML Cookbook (2nd edition)*. Estados Unidos da América, O'Reilly, 2003.

[2] GEARY, D., HORSTMANN, C. S., *Core JavaServer Faces (3rd edition)*. Estados Unidos da América, Prentice Hall, 2010.

[3] SIERRA, K., BATES, B., *Head First Java (2nd edition)*. Estados Unidos da América, O'Reilly, 2005.

[4] SCHWAB, B., *AI Game Engine Programming*. Estados Unidos da América, Charles River Media, 2004.

[5] RUSSEL, S., NORVIG, P., *Artificial Intelligence: A Modern Approach (3rd edition)*. Estados Unidos da América, Prentice Hall, 2010.

[6] KALIN, M., *Java Web Services: Up and Running*. Estados Unidos da América, O'Reilly, 2009.

[7] FREEMAN, E. T., FREEMAN, E., *Head First HTML with CSS & XHTML*. Estados Unidos da América, O'Reilly, 2006.

[8] "Background", http://ai-depot.com/FiniteStateMachines/FSM-Background.html.

[9] "Java Applet Tutorial", http://www.realapplets.com/tutorial/.

[10] "Desenvolvendo Web Services usando JAX-WS", http://www.devmedia.com.br/articles/viewcomp.asp?comp=2374.

[11] "CSS Tutorial", http://www.pt-br.html.net/tutorials/css/.

[12] "JavaScript Tutorial", http://www.w3schools.com/js/default.asp.

[13] "HTTP Tutorial", http://www.comptechdoc.org/independent/web/http/reference/.

[14] "JSON", http://www.json.org/.

[15] “Java Server Faces (JSF)”, http://www.dsc.ufcg.edu.br/~jacques/cursos/daca/html/jsf/jsf.htm.

[16] “Fundamentos de Servlets”, http://www.devmedia.com.br/articles/viewcomp.asp?comp=3573

[17] “Facelets”, http://www.devmedia.com.br/articles/viewcomp.asp?comp=5332.

[18] “O que é JSON? Como e quando usar.”, http://www.brnsouza.com/blog/index.php/2009/05/o-que-e-json-como-e-quando-usar/.

[19] “Web Services, SOAP e Aplicações Web”, http://devedge-temp.mozilla.org/viewsource/2002/soap-overview/index_pt_br.html.

[20] “Protocolo de Transporte Padrão – SOAP”, http://imasters.uol.com.br/artigo/4379/webservices/protocolo_de_transporte_padrao_soap/.

[21] “SOAP (Simple Object Access Protocol)”, http://www.gta.ufrj.br/grad/07_2/daniel/index.html.

[22] “Utilização da Linguagem de Programação JAVA com Banco de Dados Relacionais”, http://www.inf.uri.com.br/utilizacao.htm.

[23] “Acessando banco de dados em Java (PARTE 1)”, http://javafree.uol.com.br/artigo/1356/Acessando-banco-de-dados-em-Java-PARTE-1.html.

[24] “Acesso a Banco de Dados em Java (JDBC)”, http://www.inf.furb.br/~jomi/java/pdf/jdbc.pdf.

Apêndice A

Código fonte da classe DataAccess

Abaixo se encontra o código fonte da classe usada para fazer o acesso genérico ao BD.

```
package DataAccess;

import java.sql.*;

/**
 *
 * @author Eliseu
 */
public class DA {
    public static final String OK_MESSAGE = "OK";
    public static final String DB_FSZ = "jdbc:postgresql://localhost:5432/DB_FSZ";
    public static final String DB_USER = "projeto";
    public static final String DB_PWD = "projeto";
    public static final String EQUAL = "=";
    public static final String H_THAN = ">";
    public static final String L_THAN = "<";
    public static final String H_E_THAN = ">=";
    public static final String L_E_THAN = "<=";


    public Connection m_con;
    public Statement m_stm;
    private String m_SQL;
```

```
        private ResultSet m_rs;

        //Connect methods beginning
        public String connect ()
        {
           try {
              Class.forName("org.postgresql.Driver");
              m_con = DriverManager.getConnection(DA.DB_FSZ, DA.DB_USER,
DA.DB_PWD);
              return OK_MESSAGE;
           } catch (Exception ex)
           {
              return ex.toString();
           }
        }

        public String connect (String url, String user, String password)
        {
           try {
              Class.forName("org.postgresql.Driver");
              m_con = DriverManager.getConnection(url, user, password);
              return OK_MESSAGE;
           } catch (Exception ex)
           {
              return ex.toString();
           }
        }
        //Connect methods ending

        //Disconnect method beginning
        public String disconnect ()
        {
           try {
              m_con.close();
```

```
    } catch (Exception ex) {
        return ex.toString();
    }
    return OK_MESSAGE;
}
//Disconnect method ending

//All the SELECT methods above
public ResultSet select (String[] columnsToSelect, String table,
        String[] columns, String[] conditionalOperators,
        String[] conditionalValues)
{
    try {
        m_SQL = "Select ";

        if (columnsToSelect!=null)
        {
            for (int i=0; i<columnsToSelect.length; i++)
            {
                if (i!=0)
                    m_SQL+=", ";
                m_SQL+=columnsToSelect[i];
            }
        } else m_SQL+="*";
        m_SQL+=" from ";
        m_SQL+=table;

        if ((columns!=null) && (conditionalValues!=null) &&
                (conditionalOperators!=null))
        {
            if ((columns.length==conditionalValues.length) &&
                    (columns.length==conditionalOperators.length))
            {
                m_SQL+=" where ";
```

```
                    for (int i=0; i<columns.length; i++)
                    {
                       if (i!=0)
                          m_SQL+=" and ";

m_SQL+=columns[i]+conditionalOperators[i]+"'"+conditionalValues[i]+"'";
                    }
                 }
              }
              System.out.println(m_SQL);
              m_stm = m_con.createStatement();
              m_rs = m_stm.executeQuery(m_SQL);
           } catch (Exception ex)
           {
              m_rs = null;
           }

           return m_rs;
        }
        //End of the SELECT methods

        //All the INSERT methods above
        public int insert (String table, String[] values)
        {
           int ret = -1;
           try {
              if (table!=null)
              if (values!=null)
              {
                 if (values.length!=0)
                 {
                    m_SQL = "Insert into "+table+" values (";
                    for (int i=0; i<values.length; i++)
                    {
```

```
                if (i!=0)
                    m_SQL+=",";
                m_SQL+="'"+values[i]+"'";
            }
            m_SQL+=")";
            System.out.println(m_SQL);
            m_stm = m_con.createStatement();
            ret=m_stm.executeUpdate(m_SQL);
        }
      }
   } catch (Exception ex)
   {
      ret = -1;
   }

   return ret;
}
//End of the INSERT methods

//All the UPDATE methods above
public int update (String table, String[] columns, String[] values,
      String primaryKey, String primaryKeyValue)
{
   int ret = -1;
   try {
      if (table!=null)
      if ((values!=null) && (columns!=null))
      {
         if ((values.length!=0) && (values.length==columns.length))
         {
            m_SQL = "Update "+table+" set ";
            for (int i=0; i<values.length; i++)
            {
               if (i!=0)
```

```
                    m_SQL+=", ";
                m_SQL+=columns[i]+"='"+values[i]+"'";
                }
                m_SQL+= " where "+primaryKey+"='"+primaryKeyValue+"'";

                System.out.println(m_SQL);
                m_stm = m_con.createStatement();
                ret=m_stm.executeUpdate(m_SQL);
            }
          }
        } catch (Exception ex)
        {
          System.out.println(ex.toString());
          ret = -1;
        }

        return ret;
      }
      //End of the UPDATE methods

      //All the DELETE methods
      public int delete (String table, String[] columns, String[] values, String[]
operators)
      {
        int ret = -1;
        try {
          if (table!=null)
          if ((values!=null) && (columns!=null))
          {
            if ((values.length!=0) && (values.length==columns.length))
            {
              m_SQL = "Delete from "+table+ " where ";
              for (int i=0; i<values.length; i++)
              {
```

```
                if (i!=0)
                    m_SQL+=", ";
                m_SQL+=columns[i]+"='"+values[i]+"'";
            }

            System.out.println(m_SQL);
            m_stm = m_con.createStatement();
            ret=m_stm.executeUpdate(m_SQL);
          }
        }
      } catch (Exception ex)
      {
        System.out.println(ex.toString());
        ret = -1;
      }

      return ret;
    }
    //End of the DELETE methods
}
```

Apêndice B

Código fonte da classe DAOFirstnames

Segue abaixo o código fonte de uma das classes de DAOs.

```
package DAOs;

import DataAccess.DA;
import TableClasses.Firstname;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;

/**
 *
 * @author Eliseu
 */
public class DAOFirstnames {
    private DA da = new DA();
    private String DEFAULT_TABLE = Firstname.FIRSTNAMES_TABLE;

    public ArrayList<Firstname> getAll ()
    {
        ArrayList<Firstname> all = new ArrayList<Firstname>();

        if(!(da.connect().equals(DA.OK_MESSAGE)))
            return null;
        ResultSet rs = da.select(null, DEFAULT_TABLE, null, null, null);
        try {
            while (rs.next())
                all.add(fillFirstNameObject(rs));
        } catch (Exception ex)
```

```
        {
            all = null;
        }
        finally
        {
            da.disconnect();
        }

        return all;
    }

    public Firstname getByID (int id)
    {
        if (!(da.connect().equals(DA.OK_MESSAGE)))
            return null;

        String[] columns = {Firstname.FIRSTNAME_ID};
        String[] operator = {DA.EQUAL};
        String[] values = {Integer.toString(id)};

        ResultSet rs = da.select(null, DEFAULT_TABLE, columns, operator,
values);
        try {
            rs.next();
            return fillFirstNameObject(rs);
        } catch (Exception ex)
        {
            return null;
        }
        finally
        {
            da.disconnect();
        }
    }
```

```
    private Firstname fillFirstNameObject(ResultSet rs) throws SQLException
    {
        Firstname temp = new Firstname();
        temp.setFirstnameId(rs.getInt(Firstname.FIRSTNAME_ID));
        temp.setFirstname(rs.getString(Firstname.FIRSTNAME));
        return temp;
    }
}
```

Apêndice C

Código fonte da classe Firtname

Segue abaixo o código fonte de uma das classes de tabela.

```
package TableClasses;

/**
 *
 * @author Eliseu
 */
public class Firstname {
    public static String FIRSTNAMES_TABLE   = "firstnames";
    public static String FIRSTNAME_ID       = "firstname_id";
    public static String FIRSTNAME          = "firstname";

    private int m_firstnameId;
    private String m_firstname;

    public String getFirstname() {
        return m_firstname;
    }

    public void setFirstname(String m_firstname) {
        this.m_firstname = m_firstname;
    }

    public int getFirstnameId() {
        return m_firstnameId;
    }

    public void setFirstnameId(int m_firstnameId) {
```

```
        this.m_firstnameId = m_firstnameId;
    }

}
```