CENTRO UNIVERSITÁRIO FEEVALE

DIOSEF DA SILVA NIEDERAUER

GRAMÁTICA DE COMANDOS *SCOUT* NO VOLEIBOL

Novo Hamburgo, dezembro de 2009.

DIOSEF DA SILVA NIEDERAUER

GRAMÁTICA DE COMANDOS *SCOUT* NO VOLEIBOL

Centro Universitário Feevale
Instituto de Ciências Exatas e Tecnológicas
Curso de Ciência da Computação
Trabalho de Conclusão de Curso

Professor Orientador: Rodrigo Rafael Villarreal Goulart

Novo Hamburgo, dezembro de 2009.

AGRADECIMENTOS

Gostaria de agradecer a todos os que, de alguma maneira, contribuíram para a realização desse trabalho de conclusão, em especial:

Minha esposa Cátia de Oliveira Niederauer por estar sempre ao meu lado me incentivando, meus pais Sergio Gonçalves Niederauer e Ioni da Silva Niederauer pelo apoio nos diversos momentos, ao Rovani Marcelo Rech e Carlos Vanderlei Rech pela oportunidade do estudo, meu orientador Rodrigo Rafael Villarreal Goulart e ao meu filho William Niederauer.

Obrigado pela confiança.

RESUMO

Através do projeto de pesquisa "A IA entrando na quadra de vôlei: *scout* inteligente", iniciou-se a construção de um software de *scout* tradicional para monitorar e avaliar os fundamentos do voleibol. Porém, nesse software, o processo para registrar todos os eventos ligados aos fundamentos de um ponto de uma partida, dependendo da agilidade da pessoa com o teclado, pode ser extremamente lento, inviabilizando totalmente o uso desse tipo de recurso. Dessa forma, este trabalho tem como objetivo desenvolver um *parser* (analisador sintático) da gramática de leitura de entradas de fundamentos de voleibol do sistema *scout* projetado por Raimann, a fim de agilizar o processo de leitura de fundamentos.

Palavras-chave: Compiladores. *Parser. Scout.* Voleibol.

ABSTRACT

Through the research project “The AI in volleyball court: intelligent scout”, it started to build a traditional monitoring software to monitor and assess the fundamentals of volleyball. But in that software, the process to record all events related to the game point (during a game), depending on the speed of the person with the keyboard, can be extremely slow, preventing the full use of such appeal. Thus, this work aims to develop a grammar parser of reading input from the fundamentals of volleyball system designed by Raimann in order to increase the process of reading fundamentals volleyball.

Key words: Compilers. Parser. Scout. Volleyball.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	Applications Programming Interfaces
BNF	Backus–Naur Form
EBNF	Extended Backus–Naur Form
FIVB	Fédération Internationale de Volleyball
GPL	GNU General Public License
JAVACC	Java Compiler Compiler
LEX	Gerador de analisadores léxicos
LALR	LookAhead Left-Rigth
LR	Left-Rigth
SCOUT	Sistema Monitoramento de Desempenho
SCOUTER	Usuário do *Scout*
SLR	Simple Left-Rigth
UNIX	Sistema operacional
YACC	Yet Another Compiler-Compiler
PROMPT	Local para digitar linhas de comandos
TOKEN	Símbolo abstrato
SIS	Sistema Interpretador Scout

SUMÁRIO

INTRODUÇÃO

Hoje em dia, como muito anunciado, a informática é fundamental no processo de tomada de decisão, pois possibilita a obtenção de informações com uma melhor qualidade, auxiliando na atividade máxima de qualquer líder - a tomada de decisões. Esse é o momento no qual um líder demonstra toda a sua capacidade de direcionar sua equipe e sua razão de ser dentro de uma organização.

Para que um líder possa ter sucesso em uma atividade, é necessário analisar uma série de informações para poder realizar a melhor decisão. As equipes de voleibol não são diferentes, onde os técnicos estão constantemente monitorando os jogadores e seus desempenhos, a fim de decidir as melhores táticas ou estratégias de jogo. Neste contexto, é possível destacar os sistemas *scout* – sistemas para monitoramento de jogo de voleibol – que capturam e processam informações estatísticas de desempenho dos atletas da equipe e da adversária. Contudo, a maioria dos sistemas *scout* não leva em consideração o histórico do jogador de toda uma temporada. Os sistemas simplesmente repassam dados para a comissão técnica responsável, que avalia, num tempo reduzido, e toma decisões com os dados coletados. Em muitas vezes, essa comissão acaba decidindo mais no conhecimento empírico do técnico do que o fator racional real daquela situação.

Na Feevale foram desenvolvidos trabalhos relacionados à *scout,* com o objetivo automatizar o processo de monitoramento de jogos de voleibol via técnicas de Inteligência Artificial e Mineração de Dados. Essa pesquisa originou três trabalhos de conclusão de curso.

Um destes trabalhos é um software de *scout* tradicional para monitorar e avaliar os fundamentos do voleibol, tendo como base o uso de software livre, com a finalidade de auxiliar professores de Educação Física nas disciplinas de ensino de voleibol ou comissões técnicas de equipes esportivas. Outros, por sua vez, realizaram a análise e o projeto de um módulo de *data mining* para ser acoplado nesse sistema de *scout* e um sistema de posicionamento em quadra de voleibol, com o intuito de ilustrar a dinâmica (simulação) de

ataque e de defesa de um jogo, também auxiliando no ensino de voleibol nas disciplinas do curso de Educação Física.

Neste mesmo contexto apresentado, existem no mercado softwares estatísticos que auxiliam nessa tarefa de monitoramento de desempenho de atletas de voleibol. Um exemplo a citar é o Data Volley da empresa italiana DataProject. A versão de demonstração desse software pode ser adquirida pelo site do fabricante gratuitamente, porém com várias restrições no nível de configuração mais avançada, sendo necessário a compra do software. A grande maioria das seleções mundiais de voleibol utiliza o Data Volley. A seleção Brasileira comandada pelo técnico Bernardinho, por sua vez, é uma das poucas que utiliza um programa específico, criado exclusivamente para ele.

Um sistema *scout*, como já apresentado anteriormente, é um sistema estatístico que monitora uma determinada equipe em relação aos fundamentos de seus jogadores (Saque, Defesa/Passe, Levantamento, Ataque/Cortada e Bloqueio), gerando informações valiosas na tomada de decisões a nível estratégico. Entretanto, para que seja possível a entrada de dados no sistema de forma mais rápida, é necessário que esses dados estejam na forma de caracteres em formato de comandos textuais (cadeia de caracteres). No software de *scout* produzido na Feevale, o processo para registrar todos os eventos/fundamentos de um ponto da partida (que em média dura alguns segundos), dependendo da agilidade da pessoa com o teclado, pode levar em torno de dois minutos, inviabilizando totalmente o sistema.

Assim, o objetivo deste Trabalho de Conclusão de Curso é desenvolver o *parser* (analisador sintático) da gramática de leitura de entradas de fundamentos de voleibol, para promover melhores desempenhos no processo de leitura desses comandos. Utiliza-se neste projeto a metodologia científica por meio de pesquisa bibliográfica e um estudo de caso para avaliação da gramática proposta.

Finalmente, para auxiliar na compreensão desta proposta o trabalho foi dividido em quatro capítulos. No capítulo 1 aborta sobre o universo do voleibol, principais fundamentos, seus atributos, a importância de realizar monitoração em jogos e, principalmente, via sistema automatizado. O capítulo 2 trata sobre o contexto de compiladores e interpretadores, estruturas, suas funcionalidades, e as metodologias de construção desses sistemas. No capítulo 3 é discutida a proposta do trabalho, informações sobre a gramática atual, ferramentas para construção de gramáticas e o que se espera com este trabalho de conclusão. O capítulo 4 é descrita as etapas elaboradas para o desenvolvimento do protótipo proposto neste trabalho de

conclusão. Na seqüência, as conclusões finais e as referências utilizadas na construção deste texto.

1 VOLEIBOL

Neste capítulo é descrita a importância de se obter dados durante uma partida de voleibol para posteriormente analisar e propor boas táticas ou estratégias. Dessa forma, no capítulo são abordados o processo de decisão e voleibol, fundamentos do voleibol, sistemas de monitoramento.

1.1 Processo de decisão e voleibol

A tecnologia atualmente, é essencial na comunicação e no armazenamento dos dados informações e dos conhecimentos, bem como na integração dos tomadores de decisão, conduzindo a um aumento da capacidade de compartilhamento da informação e do conhecimento (JUNQUEIRA, 1998). O que acaba auxiliando na atividade máxima de qualquer líder – a tomada de decisão. Esse é o momento no qual o líder demonstra toda a sua capacidade de direcionar sua equipe e sua razão de ser dentro de uma organização (BINDER, 1994).

Para que um líder possa ter sucesso em uma atividade, é necessário analisar uma série de informações para poder realizar a melhor decisão. As equipes de voleibol não são diferentes, os técnicos estão constantemente monitorando os jogadores e seus desempenhos durante uma partida de voleibol, a fim de decidir as melhores ações (que jogada deve ser realizada e por quem; qual a posição na quadra adversária que saque deve atingir e com qual potencia; qual dos levantadores deve-se jogar; qual momento do jogo deve-se solicitar um tempo; qual jogador deve sair para a entrada do jogador líbero[1], etc.).

[1] Segundo João (2004), líbero é um jogador especialista nos fundamentos de recepção e defesa. O líbero foi criado pela FIVB em 1998, com o propósito de permitir disputas mais longas de pontos e tornar o jogo mais atraente para o público (FIVB, 2009).

1.2 Fundamentos do voleibol

Para decidir onde um jogador deve sacar ou atacar, qual a sua posição na quadra de vôlei o jogador vai oferecer o maior rendimento, que tipo de treinamento deve ser realizado para apurar um atleta e/ou toda equipe. Enfim, qual a melhor estratégica ou quais táticas deverão ser utilizadas dependem exclusivamente do monitoramento dos fundamentos do voleibol: saque, defesa/passe, levantamento, ataque/cortada e bloqueio (ZAMBERLAM, 2005). Esses fundamentos, segundo Raimann (2007), foram elencados após entrevistas com um técnico de voleibol e um professor do curso de educação física da Feevale.

Segue um detalhamento sobre esses fundamentos para auxiliar no entendimento do *scout* determinando assim quais dados são importantes e como eles são utilizados.

1.2.1 Fundamentos saque

Para o fundamento saque foram levantados os seguintes atributos:

- Número da camiseta do jogador: atributo necessário para identificar o jogador que exerce o fundamento;
- Posição do jogador no saque: atributo para determinar a posição dentro da quadra em que o jogador faz o saque;
- Tipo de saque: atributo que identifica o saque;
- Direção do saque: atributo para determinar a direção do saque;
- Resultado do saque: esse atributo é, em muitos casos, validado em relação ao atributo resultado do fundamento passe adversário.

Todos os atributos apresentados dependem de uma boa e correta interpretação do profissional que estiver utilizando o *scout*, neste estudo chamado de *scouter*. A Figura 1.1 ilustra os conceitos, atributos e relações do fundamento saque.

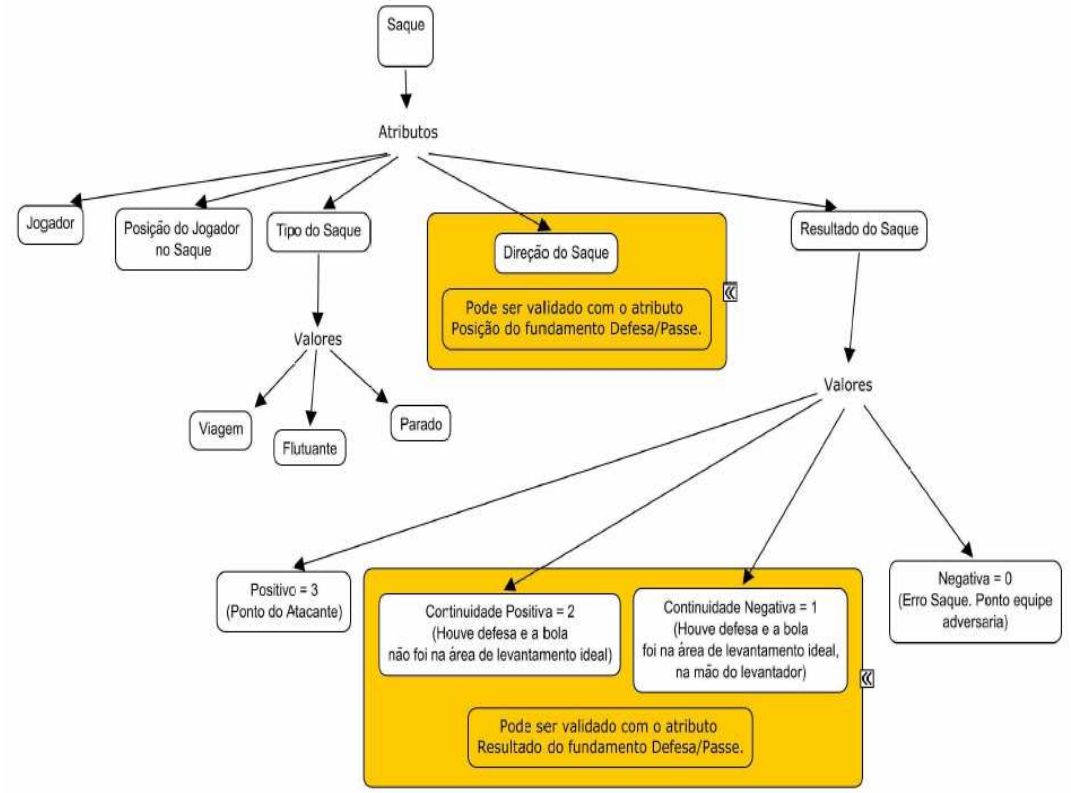

Figura 1.1 - Mapa conceitual do fundamento de saque.
Fonte: (RAIMANN, 2007)

1.2.2 Fundamentos defesa/passe

Para o fundamento de defesa/passe foram levantados os seguintes atributos:

- Número da camiseta do jogador: esse atributo é necessário em todos os fundamentos, pois não é possível afirmar que o jogador que está na posição trajeto da bola, vai receber a bola;
- Posição do jogador na defesa ou passe: esse atributo pode ser utilizado para validar os dados informados, a fim de fornecer um controle maior. Pode ser validado com o atributo direção dos fundamentos saque ou cortada;
- Resultado da defesa ou passe: esse atributo possui três valores, sendo:

  0 – Esse valor significa negativo, ou seja, a bola bateu no jogador e foi para fora da quadra, possibilitando assim um ponto para a equipe adversário;

  1 – O valor significa continuidade negativa, ou seja, o jogador fez a defesa, mas não conseguiu passar a bola para área ideal de levantamento;

2 – significa continuidade positiva, ou seja, o jogador conseguiu passar a bola para área de levantamento;

- Tipo de ocorrência do fundamento: esse atributo é utilizado para determinar se o fundamento representa uma defesa, expresso pelo caractere “d” – defesa, em que o jogador recebe uma bola vinda de uma cortada, ou representado pelo caractere “p” – passe, em que o jogador recebe uma bola vinda de um saque.

A Figura 1.2 mostra a estrutura completa dos fundamentos de defesa/passe.

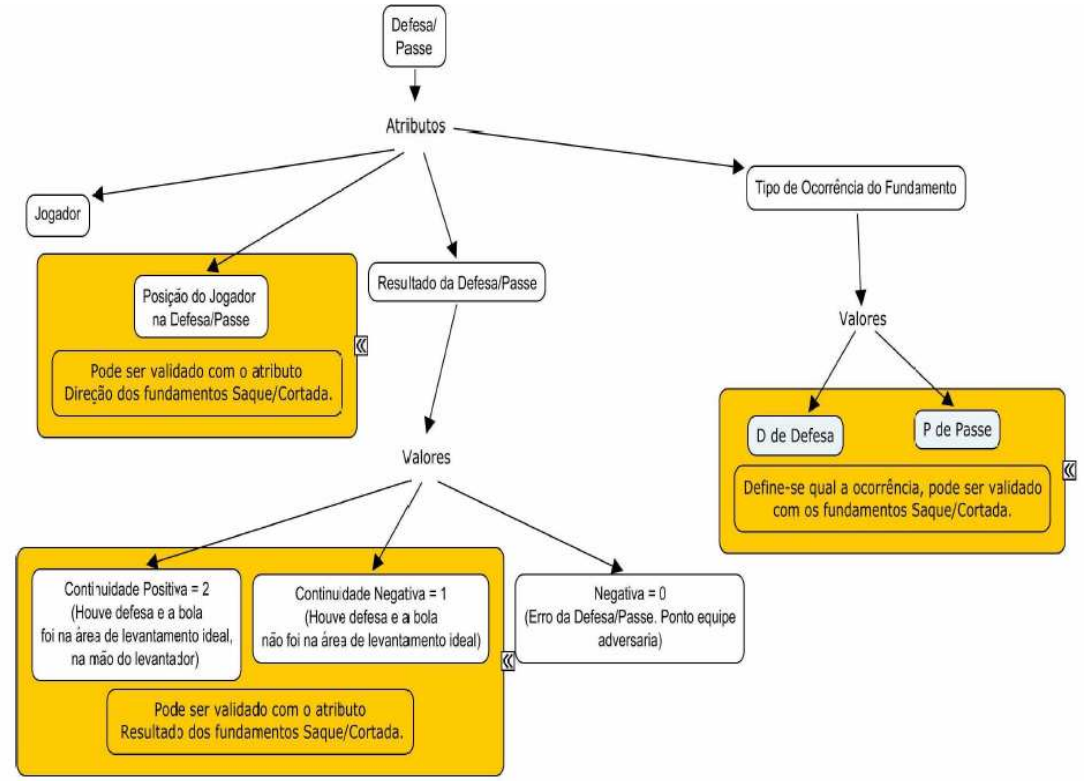

Figura 1.2 - Mapa conceitual do fundamento defesa/passe.
Fonte: (RAIMANN, 2007)

1.2.3 Fundamento levantamento

Para o fundamento de levantamento foram levantados os seguintes atributos:

- Número da camiseta do jogador;
- Posição do jogador no levantamento;
- Tipo de levantamento: esse atributo é utilizado para determinar qual tipo de levantamento foi realizado pelo jogador. O levantamento “em suspensão” é

representado pelo caractere “s” e o levantamento “no chão” representado pelo caractere “c”;

- Qualidade do levantamento: para determinar esse atributo são utilizados quatro valores, sendo:

  h – bola “chutada”;

  s – bola em “segurança”;

  m – bola de “meio”;

  e – bola “enforcada”.

- Direção do levantamento: esse atributo é utilizado para determinar para onde a bola foi lançada. O atributo pode ser validado com o atributo posição do fundamento cortada, ou seja, o jogador que irá fazer a cortada recebe a bola na mesma posição para onde foi feito o levantamento.

A Figura 1.3 mostra a estrutura completa do fundamento de levantamento.

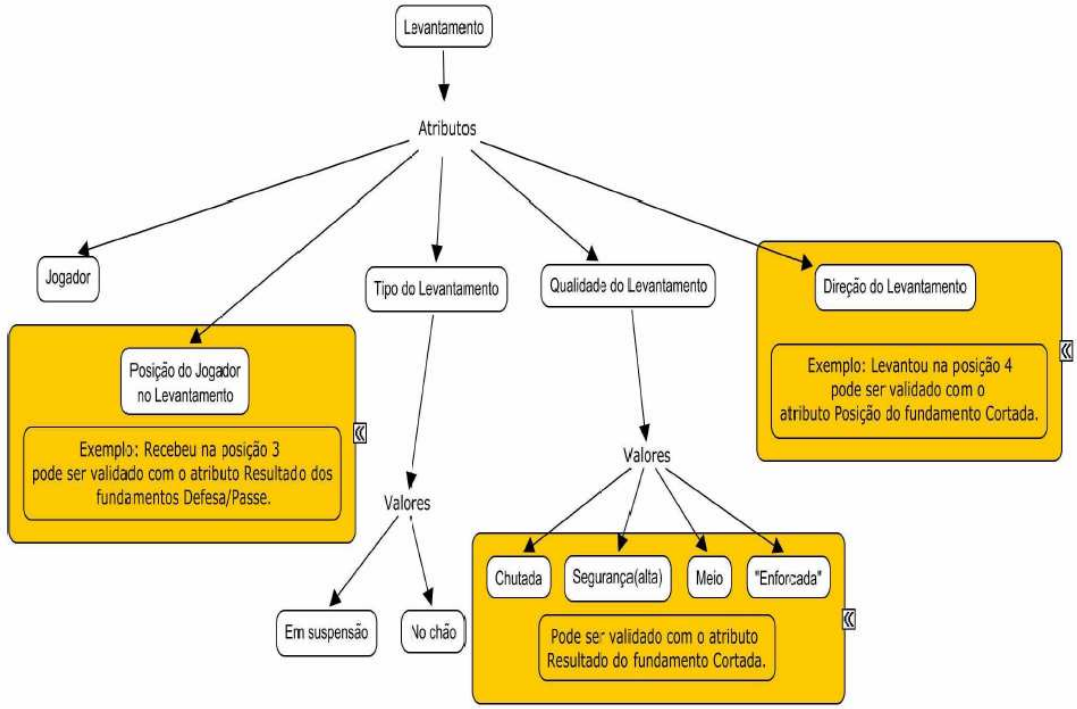

Figura 1.3 - Mapa conceitual do fundamento levantamento.
Fonte: (RAIMANN, 2007)

1.2.4 Fundamento cortada

Para o fundamento de cortada foram levantados os seguintes atributos:

- Número da camiseta do jogador;

- Posição do jogador na cortada: o atributo pode ser validado com o atributo direção do fundamento levantamento;
- Direção da cortada: o atributo pode ser validado com o atributo posição do fundamento defesa;
- Resultado da cortada: esse atributo é, em muitos casos, validado em relação ao atributo resultado do fundamento defesa do adversário. Podendo ter os seguintes valores:

  0 – errando a cortada;

  1 – fazendo uma boa recepção, na mão do levantador;

  2 – tendo dificuldade de receber a bola, não passando para o levantador;

  3 – fazendo a cortada fazendo um ponto direto.

- Velocidade da cortada: esse atributo é utilizado para contemplar jogos de vôlei de duplas de areia. Possuindo dois valores, sendo:

  r – para cortada do tipo “rápida”;

  p – para corada do tipo “pingada”.

A Figura 1.4 mostra a estrutura completa do fundamento cortada.

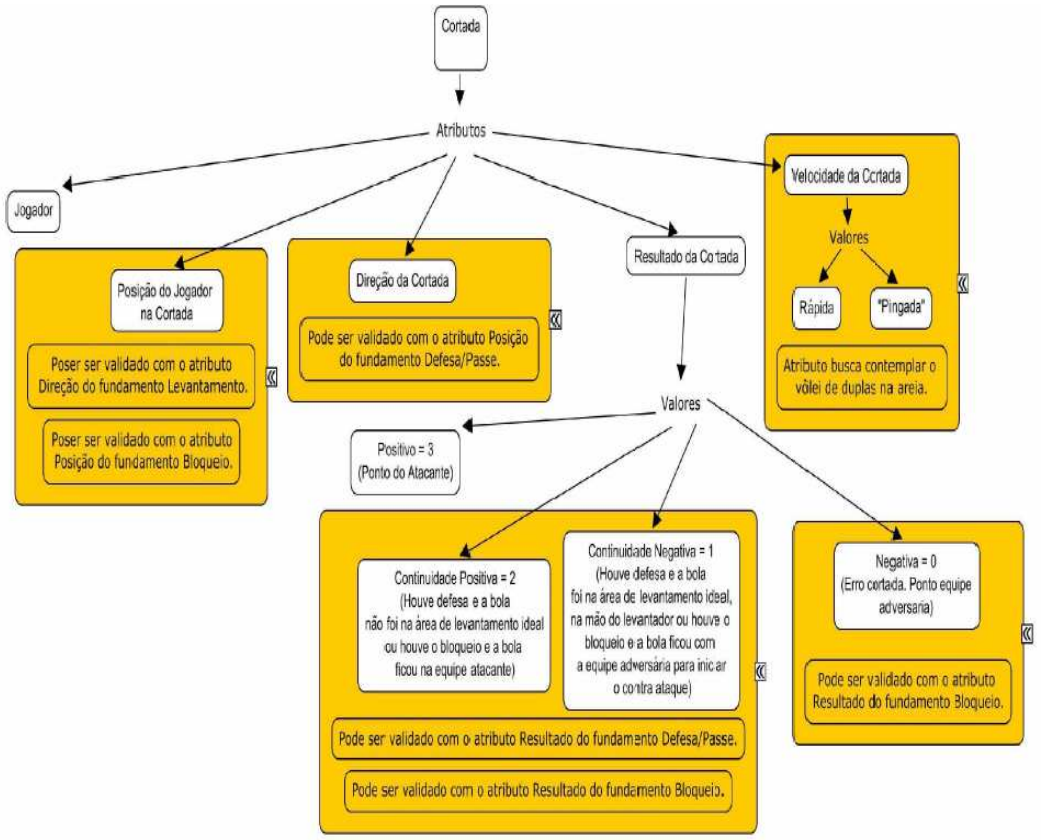

Figura 1.4 - Mapa conceitual do fundamento cortada.
Fonte: (RAIMANN, 2007)

1.2.5 Fundamento bloqueio

Para o fundamento de bloqueio foram levantados os seguintes atributos:

- Número da camiseta do jogador;
- Posição do jogador no bloqueio: o fundamento pode ser validado com o atributo posição do fundamento cortada;
- Constituição do bloqueio: esse fundamento é utilizado para determinar o número de jogadores que fazem parte do bloqueio. Possuindo os seguintes valores:

  s – bloqueio simples (único jogador);

  d – bloqueio duplo;

  t – bloqueio triplo.

- Resultado do bloqueio: esse fundamento segue a lógica do resultado dos fundamentos anteriores. Possuindo os seguintes valores:

0 – tocado no bloqueio e indo para fora (ponto equipe atacante);

1 – tocado no bloqueio e voltado para a equipe atacante;

2 – bloqueio amorteceu a bola (contra ataque);

3 – bloqueio tenha feito ponto.

A Figura 1.5 mostra a estrutura completa do fundamento bloqueio.

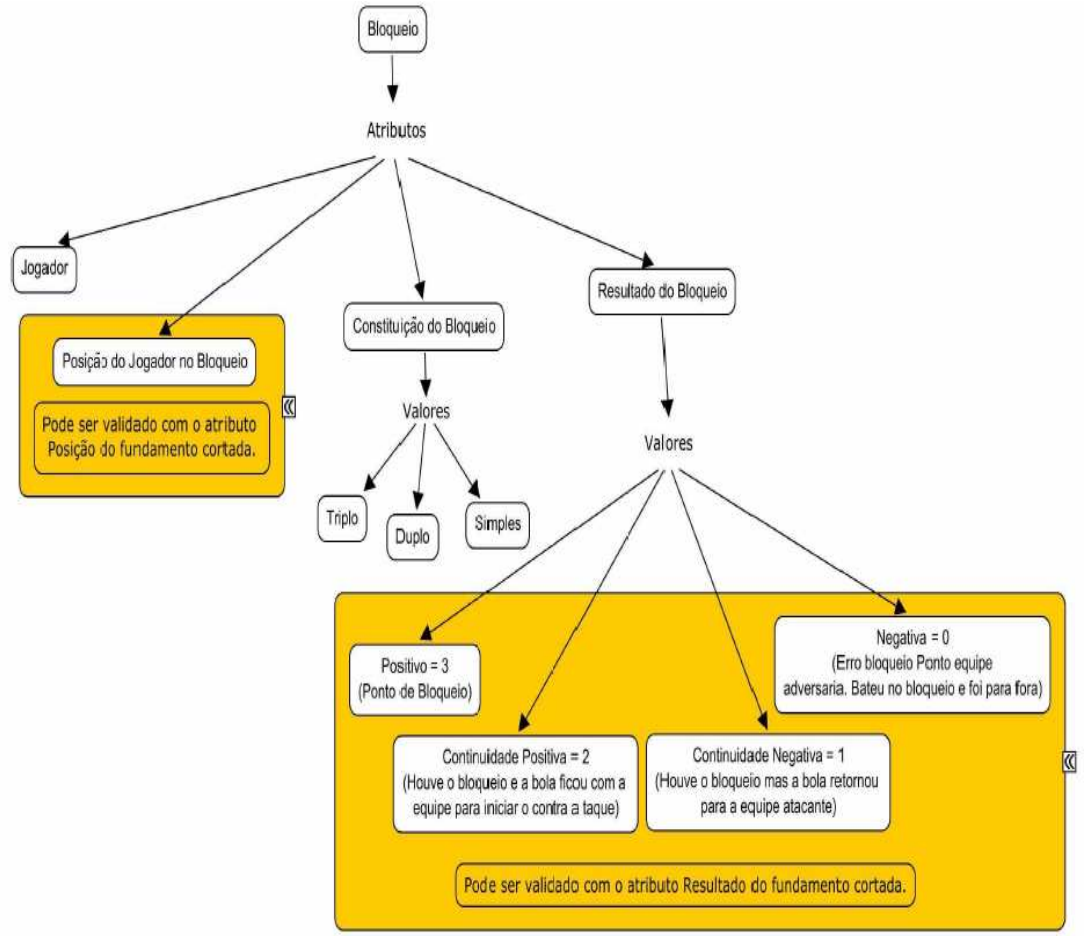

Figura 1.5 - Mapa conceitual do fundamento bloqueio.
Fonte: (RAIMANN, 2007)

1.2.6 Relação do efeito entre os fundamentos

Para facilitar a compreensão dos requisitos dos fundamentos apresentados (saque, defesa ou passe, levantamento cortada e bloqueio) Raimann (2007) representou na forma de desenho os atributos necessários em cada fundamento. Na Figura 1.6 o mapa conceitual da relação dos efeitos entre os fundamentos anteriormente relatados.

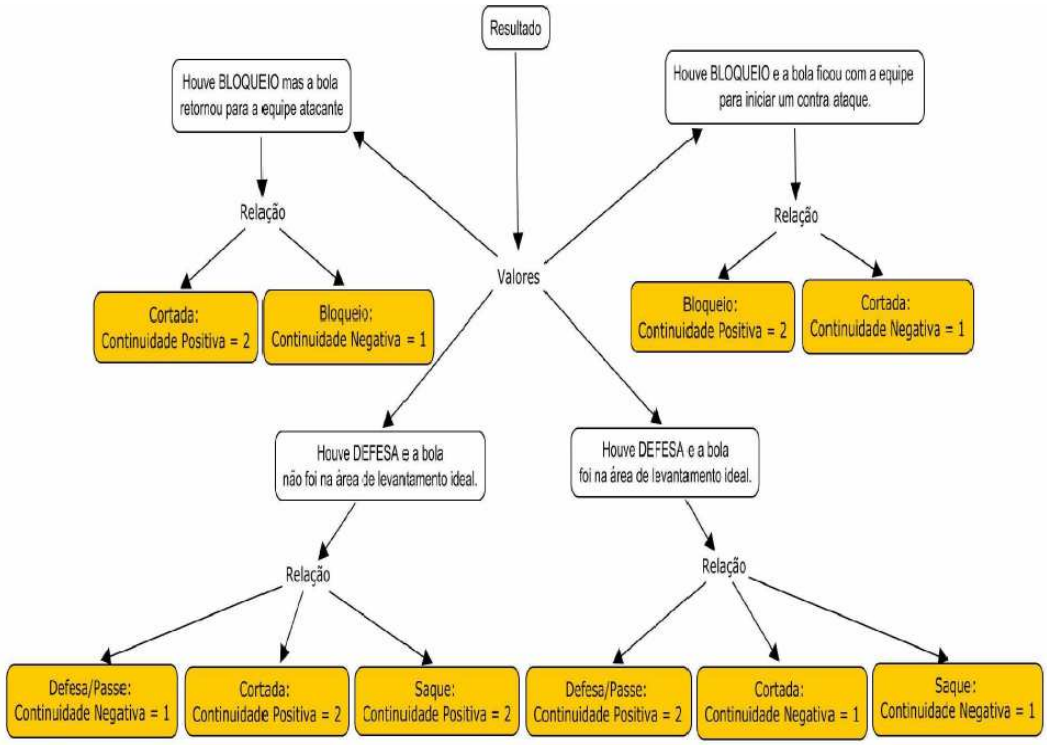

Figura 1.6 - Mapa conceitual da relação do efeito entre os fundamentos.
Fonte: (RAIMANN, 2007)

1.3 Sistemas de monitoramento de voleibol

Sistemas de monitoramento de voleibol, também conhecidos como sistemas *scout,* são softwares que capturam e processam informações estatísticas de desempenho dos atletas da equipe e da adversária (ZAMBERLAM, 2005).

Os sistemas *scout* podem ser divididos em *scout* técnico e *scout* tático. De acordo Roberta Giglio (BALIEIRO, 2004), *scout* técnico seve para o treinador avaliar o desempenho de sua própria equipe, levando em consideração os principais fundamentos do voleibol. Já o *scout* tático faz um mapeamento da quantidade, do percentual e dos tipos de jogadores do time adversário.

A seleção Brasileira do técnico Bernardinho é uma das poucas que utiliza um software especifico para o *scout* técnico e outro para o *scout* tático, criados exclusivamente para ele. Dessa forma a comissão técnica não fica a mercê de um fabricante, atendendo as necessidades da equipe, além de fácil manutenção sendo bem flexível (BERNARDINHO, 2006).

Os sistemas de *scout* apresentam os seguintes aspectos estruturais: Telas cadastrais (Atletas, Adversário, Posições, Competições, Categorias, Equipes). Podem ser softwares livres ou proprietários, esses últimos, na grande maioria possuem preços bem elevados.

Os aspectos funcionais desses sistemas são: Gestão de time, Histórico de partidas, Relatórios individualizados (estatísticos) de jogo, jogador e função do jogador, alguns são projetados para utilização via Web. Podem ser adaptáveis a outros esportes e dependendo podem cruzar vídeos com os dados estatísticos.

No Quadro 1.1 é apresentado um comparativo dos principais atributos dos sistemas *scout*. (**OK** indica que o sistema possui e **NA** não possui)

	Scout Graph 1.0	Sis Volei	Data Volley	Scout Raimann
Scout técnico	OK	OK	OK	OK
Scout tático	OK	OK	OK	NA
Relatórios	OK	OK	OK	OK
Web	NA	OK	NA	NA
Outros esportes	OK	NA	NA	NA
Filme do jogo	NA	NA	OK	NA
Proprietário	OK	OK	OK	NA
Livre	NA	NA	NA	OK

Quadro 1.1 - Atributos *scout*.
Fonte: autor

Finalizando o contexto de voleibol (fundamentos, sistemas de monitoramento, etc.) é necessário abordar também o tema compiladores/interpretadores, pois são através deles que são construídos os sistemas que fazem as traduções dos comandos do *scout* para que estes sejam interpretados e transformados em dados do voleibol.

2 COMPILADORES/INTERPRETADORES

Neste capítulo é tratado o contexto de compiladores e interpretadores. Como são estruturados, suas funcionalidades, bem como as metodologias de construção desses sistemas.

2.1 Compilador

O termo compilador, segundo Rangel (2009), faz referência ao processo de composição de um programa pela reunião de várias rotinas de bibliotecas e ao processo de tradução, considerado hoje função central de um compilador.

Segundo Aho (1995), compilador é programa que, a partir de um código escrito em uma determinada linguagem cria um programa semanticamente equivalente, porém escrito em outra linguagem.

Nesse processo de tradução, existem duas tarefas básicas a serem executadas por um compilador:

- Análise - onde o texto de entrada é examinado, verificado e compreendido;
- Síntese - em que o texto de saída é gerado, de forma a corresponder ao texto de entrada (AHO , 1995);

Na Figura 2.1 é detalhada esse processo de tradução.

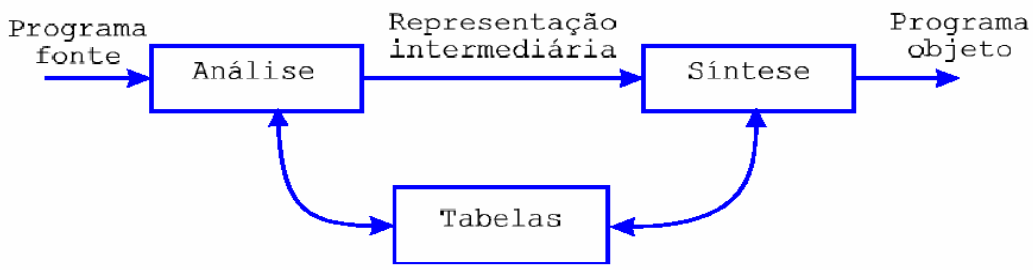

Figura 2.1 - Tabela de tradução.
Fonte: (RANGEL, 2009)

De acordo com Rangel (2009) e Louden (2009), basicamente a análise pode ser subdividida da seguinte forma:

- Analisador léxico - possui a função de separação e identificação dos elementos componentes do programa fonte;
- Analisador sintático - possui a função de determinar a estrutura ou a sintaxe de um programa, descrita através de uma linguagem de programação;
- Analise semântica - tem a função de tratar os aspectos sensíveis ao contexto da sintaxe das linguagens de programação.

Tabela de símbolos é uma estrutura de dados gerada pelo compilador, o qual contém um registro para cada identificador de variáveis, de parâmetros, de funções, de procedimentos, etc., definidos no programa fonte (AHO, 1995).

Segundo Rangel (2009), otimização, tratamento, gerenciamento ou recuperação de erros, possuem a função de diagnosticar erros léxicos, sintáticos e semânticos encontrados na etapa de análise, devendo tratar os erros encontrados, de forma que análise possa ser concluída completamente.

Principais características do gerenciamento de erros:

- Relatar erros e recuperá-los;
- Relatar erros assim que possível;
- Exibir mensagens de erros adequadas;
- Continuar mesmo após o erro;
- Evitar a cascata de erros.

Conceitualmente, um compilador opera por partes, onde cada uma transforma o programa fonte de uma representação para outra (AHO, 1995). Na Figura 2.2 é detalhada uma decomposição típica de um compilador, em que as três primeiras partes (analisador léxico, analisador sintático e analisador semântico) formam o núcleo da parte de análise do compilador e a tabela de símbolos e recuperação de erros interagem com o compilador.

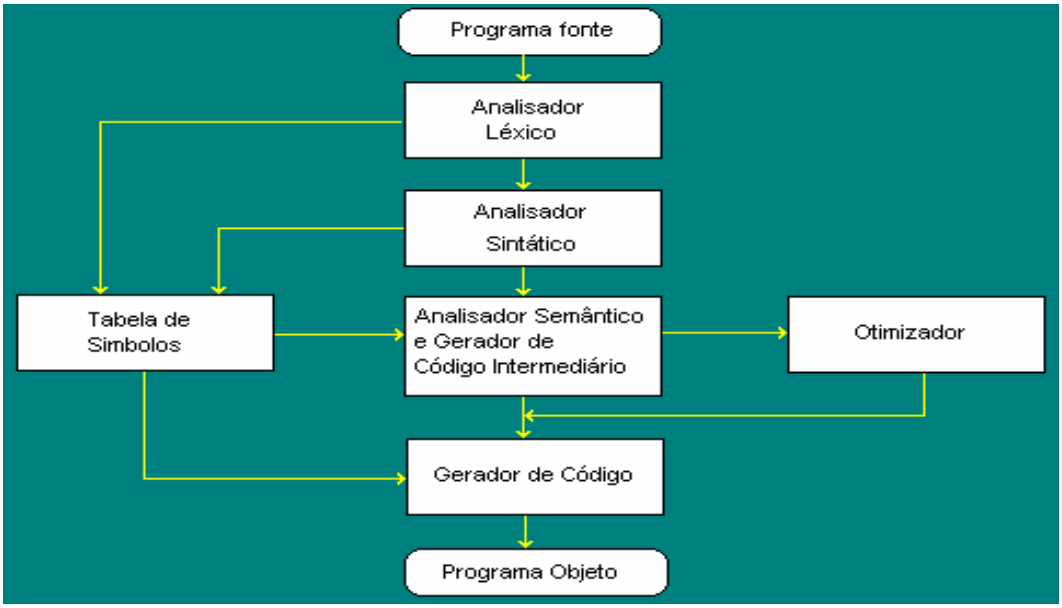

Figura 2.2 - Esquema do processo de compilação.
Fonte: (RANGEL, 2009)

2.2 Interpretadores

Certos tipos de tradutores transformam uma linguagem de programação em uma linguagem simplificada, que pode ser chamada de um código intermediário, que pode ser diretamente executado por um programa chamado interpretador.

Interpretador é um programa que interpreta diretamente as instruções do programa fonte, gerando desta forma um resultado. Dependendo da situação e da linguagem, pode ser preferível interpretar ao invés de compilar. Veja na Figura 2.3 a ilustração.

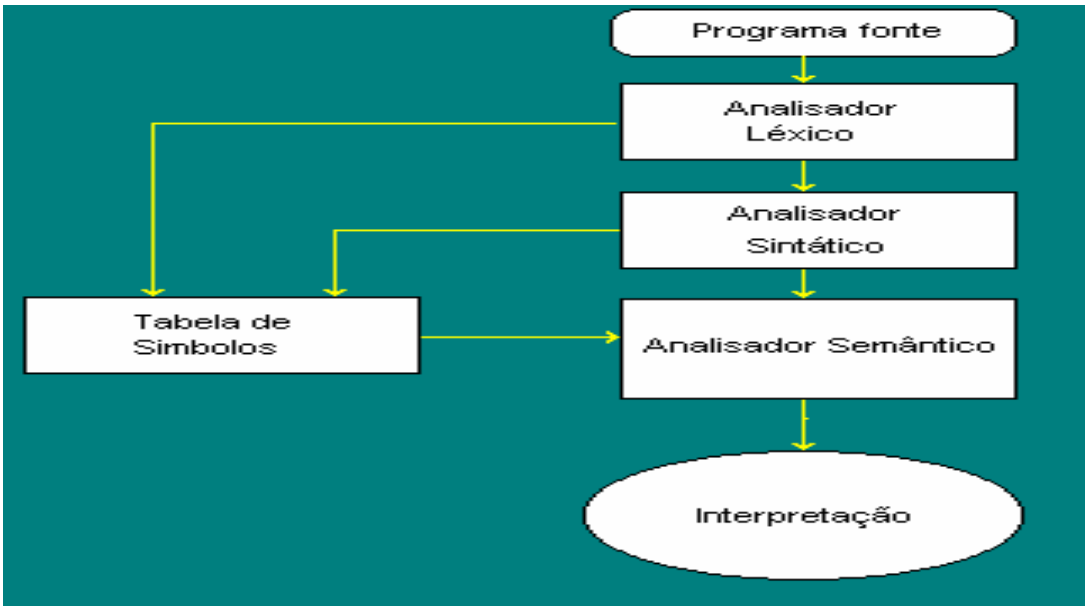

Figura 2.3 - Interpretador
Fonte: (RANGEL, 2009)

Os interpretadores são menores que os compiladores, o que acaba facilitando a implementação de construções complexas de linguagem de programação simplificada.

Segundo Guimarães (2007) há diversos modos de interpretação, a citar:

- O interpretador lê o texto do programa e vai executando as instruções uma a uma (atualmente esta forma é raríssima);
- O interpretador toma o texto do programa e o traduz para uma estrutura de dados interna, percorrendo a estrutura e interpretando o programa (após a tradução do programa para a estrutura de dados);
- Um compilador traduz o texto do programa para instruções de uma máquina virtual (pseudo-código). Essa máquina virtual é usualmente uma máquina não só simples como feita sob medida para a linguagem que se quer utilizar.

2.3 Programas relacionados a compiladores

A entrada para o compilador pode ser produzida por um ou mais pré-processadores e pode ser necessário processamento posterior da saída do compilador, antes do código de maquina ser obtido (AHO, 1995). A saber:

- Interpretadores;
- Montadores;
- Ligadores;
- Carregadores;
- Pré-processadores;
- Editores.

Para maiores detalhes, sugere-se a leitura de Aho (1995).

2.4 Parser

Parser (analisador sintático) é um algoritmo, baseado em uma gramática, capaz de construir uma derivação para qualquer sentença em alguma linguagem (AHO, 1995).

Também segundo Aho (1995), é possível definir gramática como um conjunto de regras de formação que definem de maneira rigorosa o modo de geração de textos corretos de uma determinada linguagem.

O *parser* é parte integrante de um compilador ou de um interpretador. Porém o enquanto o compilador possui no processo de tradução as tarefas de análise e síntese, o interpretador possui somente a parte de análise. Para maiores detalhes, sugere-se a leitura de Aho (1995).

A gramática que o *parser* irá analisar é a de leitura de entradas de fundamentos de voleibol do sistema *scout* Raimann (2007).

2.4.1 Metodologias

Segundo escrito em (AHO, 1995; RANGEL, 2009), para o projeto de um *parser*, existem algumas metodologias. Para melhor entendimento deve-se imaginar uma "árvore gramatical" (regras de símbolos terminais e não terminais), na Figura 2.4 é apresentado um exemplo de árvore gramatical, onde < a,b,c,d, e > são os símbolos terminais e < F, G > os símbolos não terminais.

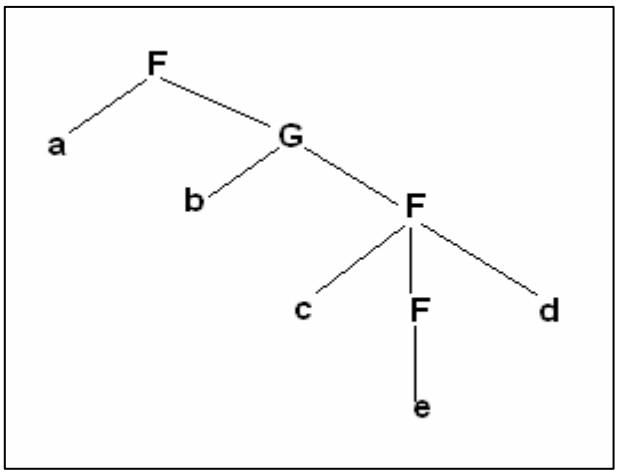

Figura 2.4 - Árvore gramatical.
Fonte: autor

Metodologias, a saber:

- Método sintático descendente (*top-down*), em que o reconhecimento da gramática é feito por expansão de regras sintáticas, substituindo símbolos não-terminais do lado esquerdo de produções pelo lado direito das produções. Na analogia da "árvore gramatical", a construção inicia na raiz e prossegue em direção às folhas. Principais tipos de *parser top-down*:
    - Recursivo com retrocesso;
    - Recursivo preditivo;

- Tabular preditivo.

- Método sintático ascendente (*bottom-up*), conhecido como análise de empilhar e deduzir. Na analogia da "árvore gramatical", a construção inicia nas folhas (o fundo) e continua até a raiz (o topo). Como as operações primárias do analisador é empilhar e reduzir existe quatro ações possíveis que o mesmo pode realizar: Empilhar, Reduzir, Aceitar e Erro.

    - Empilhar: Esta ação ocorre a partir de uma transição gerada por um item não completo, onde o número do novo estado é empilhado juntamente com o elemento lido. Somente ocorre um erro se houver um elemento não esperado na cadeia de entrada.

    - Reduzir: Esta ação ocorre a partir de um item completo no estado, todos os elementos presentes na parte direita da regra de produção são retirados da pilha e é empilhado o não terminal do item, juntamente com o novo estado.

Nesses dois métodos, a varredura léxica é feita da esquerda para a direita, símbolo a símbolo. De acordo com Aho (1995). Existem diversas técnicas da análise sintática, a saber: Análise Sintática Descendente; Análise Sintática Preditiva não Recursiva; Análise Sintática de Precedência de Operadores; Análise Sintática SLR; Análise Sintática LALR e Análise Sintática LR Canônico.

2.4.2 Ferramentas

Para o projeto de um *parser*, existem algumas ferramentas que auxiliam na construção de compiladores e/ou interpretadores, cita-se:

- Yacc (Unix / C);
- Bison (GNU / C, C++),
- JavaCC (Java).

Os sistemas de monitoramento (*scout*), em geral, trabalham com o conceito de analisar comandos textuais, que são características de interpretadores e compiladores. Segundo Raimann (2007) é possível destacar os seguintes softwares de *scout* que trabalham dessa forma: Scout Graph 1.0, SisVolei, Data Volley e o próprio *scout* desenvolvido por Raimann (2007) e utilizado nesse Trabalho de Conclusão.

2.5 Metodologias do projeto

A metodologia escolhida para a implementação do projeto do *parser* (gramática de leitura de fundamentos de voleibol) foi o método sintático descendente (*top-down*). A principal justificativa da utilização desse método foi pelo fato que o JavaCC utiliza esse método gerando código-fonte de classes Java que implementam os analisadores léxico e sintático de uma linguagem. Além disso, é o método utilizado no Compilador Verto, também estudado neste trabalho.

Finalizando sobre o capítulo de Compiladores/Interpretadores, segue uma descrição inicial sobre a proposta do Trabalho de Conclusão.

3 PROPOSTA DO TRABALHO

Este trabalho tem como objetivo desenvolver um analisador sintático da gramática de leitura de entradas de fundamentos de voleibol do sistema *scout* projetado por Raimann (2007). Nesse processo de entrada de fundamentos, deve ser possível parametrizar quais os fundamentos que serão registrados. Além disso, ser for o caso, omitir ou ignorar fundamentos que não forem possíveis registrar, otimizando o processo de leitura desses comandos. Isso, pois na proposta de Raimann (2007) se não forem informados todos os fundamentos, não é registrada a jogada. Dessa forma, no capítulo, são apresentadas informações sobre a gramática atual, ferramentas para construção de gramáticas e a proposta do trabalho.

Na Figura 3.1 apresenta o projeto completo, onde o módulo de dados estatísticos (Analisador) corresponde à proposta deste trabalho de conclusão.

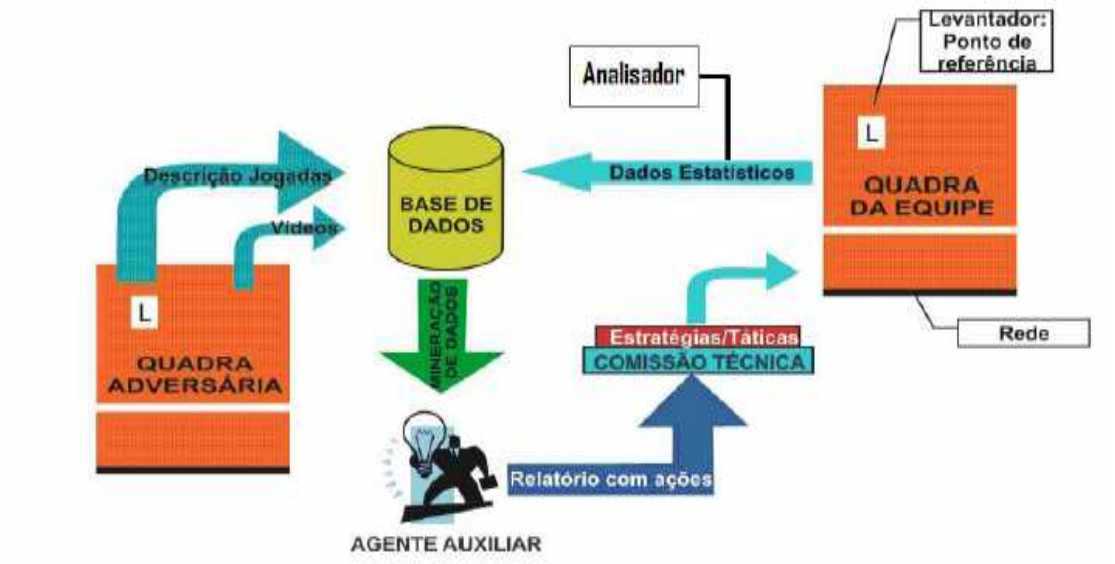

Figura 3.1 - Esquema do *Scout* inteligente.
Fonte: (ZAMBERLAM, 2005)

3.1 Gramática atual

Para o desenvolvimento do *scout* Raimann (2007), onde os fundamentos são cadastrados em forma de caracteres, foi desenvolvida uma gramática para análise de comandos. Esta gramática refere-se à entrada de dados do jogo/partida (fundamentos), segue a ordem que é registrada os fundamentos[2], separados por ponto e vírgula (;):

- Saque;
- Defesa/passe;
- Levantamento;
- Cortada;
- Bloqueio;

Na Figura 3.2, é detalhada a gramática para análise de comandos.

[2] Veja maiores detalhes no capítulo 1, seção 1.2.

```
<partida>:= <inicialização> <sets> <fim_partida>
<inicializacao>:= <equipea> <equipeb> <inicio>
<equipea>:= 'a' <jogadores> ';'
<jogadores>:= <jogador> <jogadores> | <jogador>
<jogador>:= 'p'< posição_quadra > 'c' <numero_camiseta>
<posição_quadra>:= 1 | 2 | 3 | 4 | 5 | 6
<numero_camiseta>:= 1..99
<equipeb>:= 'b' <jogadores> ';'
<inicio>:= 'b' <equipe> ';'
<equipe>:= 'a' | 'b';
<sets>:= <set> <sets> | <set>
<set>:= 's' <numero_set> ';' <eventos> 'fs' <numero_set> ';'
<numero_set>:= 1 | 2 | 3 | 4 | 5
<eventos>:= <evento> '.' <eventos> | <evento> '.'
<evento>:= <substituições> | <ponto> | <jogada> | <voltar_jogada>
// Forma para declarar um ponto: pta significa que a equipe A
// ganhou um ponto. De uma forma mais simples.
<ponto>:= 'pt'<equipe>
<voltar_jogada>:= 'vj'
<substituições>:= <substituição>';'<substituições>|<substituição>';'
<substituição>:=  's'<equipe>'c'<numero_camiseta>'c'<numero_camiseta>
<jogada>:= <lances>
<lances>:= <lance> <lances> | <lance>
<lance>:= <saque> ';'
      | <recepção> ';'
      | <levantamento> ';'
      | <cortada> ';'
      | <bloqueio> ';'
<saque>:=
'c'<numero_camiseta>'p'<posição_quadra><tipo_saque>'p'<posição_quadra>
'e'<efeito><bola_lado>
<tipo_saque>:= 'f' | 'p' | 'v'
<efeito>:= 0 | 1 | 2 | 3
// quero representar aqui que a bola não mudou de lado na quadra '' ou
mudou de lado 'x'
<bola_lado>:= '' | 'x'
<recepção>:= 'c' <numero_camiseta> 'p' <posição_quadra> 'e'
<efeito_defesa> <ocorrencia_defesa> <bola_lado>
<efeito_defesa>:= 0 | 1 | 2
<ocorrencia_defesa> := 'd' | 'p'
<levantamento>:= 'c' <numero_camiseta> 'p' <posição_quadra>
<tipo_levantada> <qualidade_levantada> 'p' <posição_quadra><bola_lado>
<tipo_levantada>:= 's' | 'c'
<qualidade_levantada>:= 'h' | 's' | 'm' | 'e'
<cortada>:= 'c' <numero_camiseta> 'p' <posição_quadra> 'p'
<posição_quadra> 'e' <efeito> <velo_cortada> <bola_lado>
<velo_cortada>:= 'r' | 'p'
<bloqueio>:= 'c' <numero_camiseta> 'p' <posição_quadra>
<tipo_bloqueio> 'e' <efeito> <bola_lado>
<tipo_bloqueio>:= 't' | 'd' | 's'
```

Figura 3.2 - Gramática atual do *Scout* Raimman.
Fonte: (RAIMANN, 2007)

A gramática proposta inicialmente por Raimann (2007) apresenta uma contribuição para a operacionalização do sistema *scout*, porém, ela tem estrutura e funcionalidades estáticas. Ou seja, se o usuário do *scout (scouter*) não estiver muito bem ambientado com os comandos, é bem provável que o sistema seja ineficiente, uma vez que há uma seqüência fixa e inflexível de comandos a serem digitados.

3.2 Ferramentas para construção de gramáticas

Conforme citado na seção 2.4.2, existem ferramentas que auxiliam na construção de compiladores e/ou interpretadores. Durante a pesquisa realizada para este trabalho, as que mais se sobressaíram foram o JavaCC[3] e as técnicas utilizadas pelo Compilador Educativo Verto (SCHNEIDER, 2005).

3.2.1 JavaCC

Segundo Deitel (2001) Java é uma linguagem de programação orientada a objetos, desenvolvida pela Sun Microsystems no inicio da década de 90, baseada nas linguagens C e C++. O Java foi projetado com o objetivo de ser portável em diversos sistemas operacionais e possui forte suporte para técnicas adequadas de engenharia de software. Esta linguagem dispõe de uma biblioteca de classes diversificada, chamada de Java API.

O JavaCC (JAVA.NET, 2009) é um gerador de analisadores sintáticos para a linguagem Java. O JavaCC é semelhante ao Yacc[4], pois gera um analisador a partir de uma gramática em notação EBNF[5], gerando como resultado código Java. O JavaCC produz o código-fonte de algumas classes Java que implementam os analisadores léxico e sintático para aquela linguagem (método *top-down*).

3.2.2 Verto

O compilador Educativo Verto surgiu da necessidade de desenvolver uma ferramenta para apoio pedagógico na disciplina de Compiladores do Centro Universitário Feevale. Ele foi escrito na linguagem de programação Java e elaborado na forma de um software livre com licença GPL, utilizando o método *top-down.* O Verto não é uma ferramenta de construção de

---

[3] https://javacc.dev.java.net/
[4] O yacc completa com o lex o conjunto de ferramentas do UNIX para a construção de compiladores.
[5] EBNF é uma gramática ampliada, formada por um conjunto finito de regras visando definir uma linguagem formal.

gramáticas, mais possui técnicas interessantes e muito bem adequadas para a construção delas. Está sendo referenciado, pois foi projetado na mesma linguagem do *scout* do trabalho e foi totalmente construído baseando-se na filosofia de software livre, que também é um dos objetivos deste projeto.

O processo de compilação do Verto dá-se em duas etapas distintas: gerando um código intermediário (formato *macro-assembler*) e gerando um código final (formato da máquina hipotética César), permitindo a execução e análise do algoritmo (SCHNEIDER, 2005).

Na Figura 3.3 é exibida a janela de edição de textos-fonte escritos na linguagem Verto.

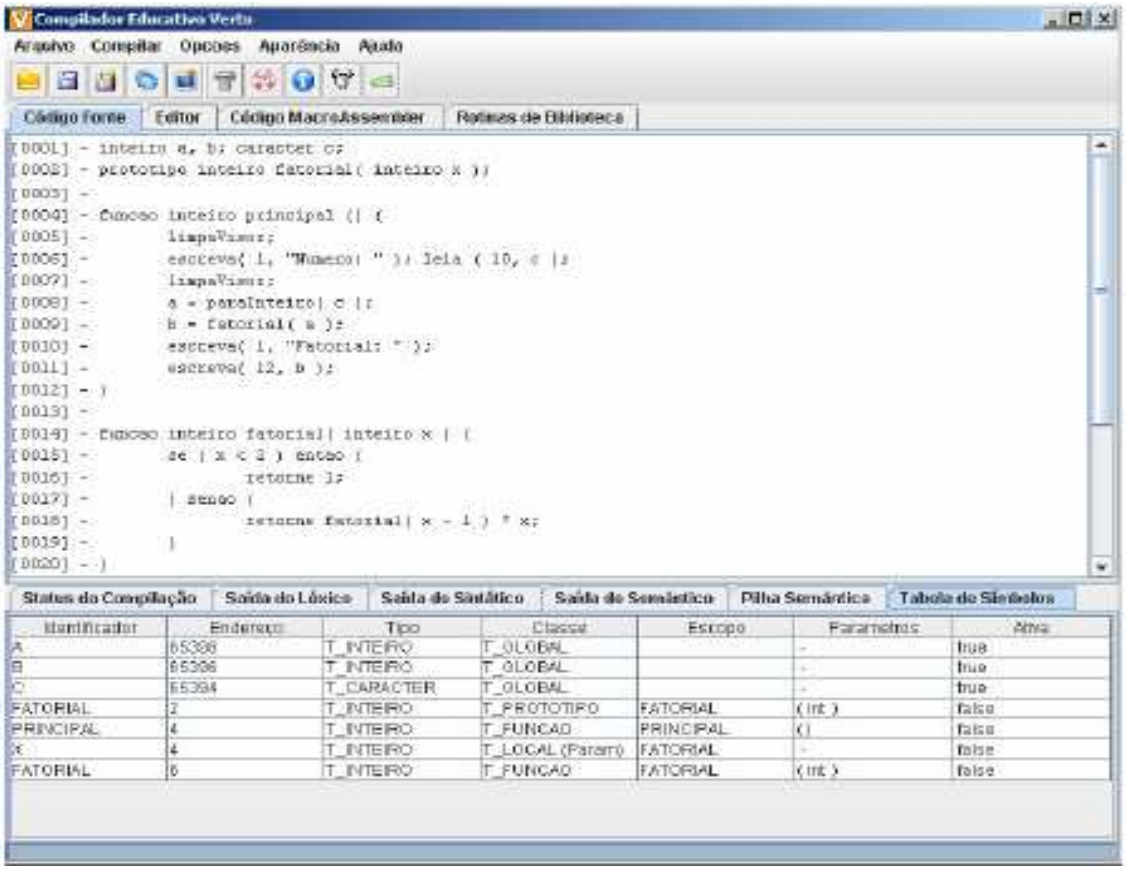

Figura 3.3 - Tela de edição do Compilador Verto.
Fonte: (SCHNEIDER, 2005)

3.3 Proposta

Uma das propostas deste trabalho de conclusão, como já apresentado, através da metodologia científica por meio de pesquisa bibliográfica e um estudo de caso é modificar a gramática inicial para análise de comandos sugerida por Raimann (2007). Pois o processo

para registrar todos os eventos/fundamentos de um ponto da partida (que em média leva alguns segundos), dependendo da agilidade da pessoa com o teclado, pode durar em torno de dois minutos, o que acaba inviabilizando totalmente o sistema.

Outra proposta sugerida é desenvolver uma interface para poder registrar os fundamentos do voleibol. Nessa interface, deve ser possível parametrizar (através de menu) quais os fundamentos serão registrados durante uma partida, fazendo com que o sistema possa (se for o caso) gerar dados de somente um tipo de fundamento, como por exemplo os fundamentos de levantamento. Pode também, se for o caso, omitir ou ignorar fundamentos que não foram possíveis registrar durante uma partida/jogo, através de uma tecla (ou teclas) pré-definida.

A seguir, é demonstrada a especificação inicial da gramática proposta para este trabalho de conclusão. A especificação foi dividida em três partes: esquema de uma gramática, símbolos e regras de produção.

Segundo Aho (1995) as gramáticas são capazes de descrever a maioria, mas não a totalidade das sínteses das linguagens de programação, porém uma parte limitada da análise sintática é realizada pelo analisador léxico, na medida em que produz uma seqüência de *tokens*[6] a partir dos caracteres de entrada.

Para a especificação de um símbolo e regras de produção é necessário separar em quatro seções distintas:

- USES – Declaração de todos os símbolos não-terminais, ou terminais de classe, usados na estrutura do símbolo;
- STRUCTURE – Discrição das diferentes formas sintáticas do símbolo (produções);
- SEMANTICS – Esta parte se subdivide em duas partes, a primeira parte constitui a declaração dos atributos herdados e sintetizados do símbolo e a segunda parte é usada para associar as regras semânticas e as condições de contexto a cada produção;
- TRANSLATION – Esta parte é opcional, serve para associar a cada forma sintática as ações necessárias para processar o símbolo definido.

---

[6] Token é um símbolo abstrato representando um tipo de unidade léxica (AHO, 1995).

Os símbolos do vocabulário inicial da gramática contêm conjuntos de terminais (*tokens*) e não terminais (símbolos que podem ser substituídos). Na Figura 3.4 foi definida alguns símbolos iniciais.

```
//
//Inicio da partida (não terminais)
//
<JOGO>

//
//Fundamentos de voleibol (terminais)
//
<SAQUE>
// Representa o fundamento de Defesa/passe
<RECEPCAO>
<LEVANTAMENTO>
<CORTADA>
<BLOQUEIO>
```

Figura 3.4 - Símbolos iniciais.
Fonte: autor

Para a criação de regras de produção, podem existir símbolos terminais e não-terminais, estabelecendo assim, uma estrutura definida da linguagem. Inicialmente foram definidas as regras dos fundamentos do voleibol, utilizando uma árvore de análise hierárquica para demonstrar essas regras. Na Figura 3.5, é representada as regras de produção dos fundamentos do voleibol, os atributos por enquanto serão os mesmos definidos por Raimann (2007), porém entre cada atributo será possível pular o fundamento que está registrando (caracterizado pelo símbolo "*").

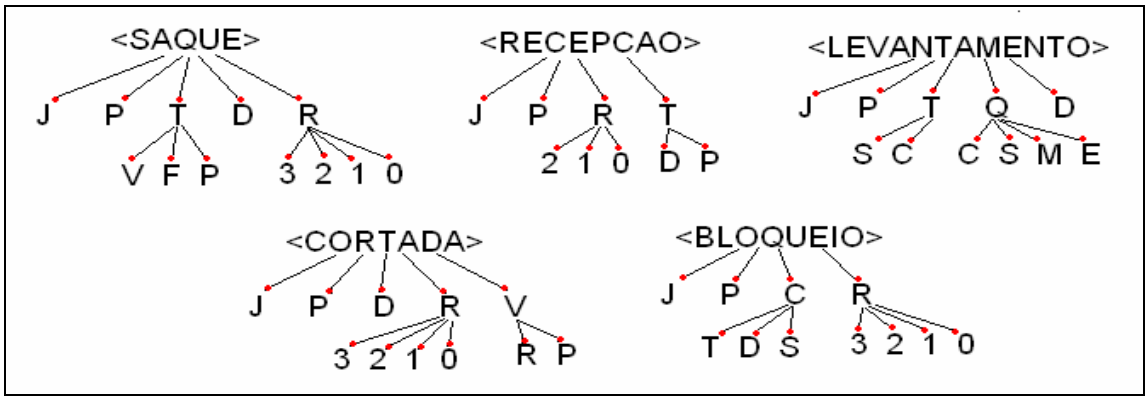

Figura 3.5 - Regras dos fundamentos do voleibol.
Fonte: autor

Como o *scouter* poderá cadastrar as teclas que preferir para configurar os fundamentos e atributos trabalhados, a gramática deve estar preparada para esta

funcionabilidade. Inicialmente nos fundamentos foi definida somente uma forma de pular os comandos, ficando como continuidade deste trabalho a inserção desta parametrização na gramática.

4 GRAMÁTICA *SCOUT*

Neste capítulo é descrita as etapas elaboradas para o desenvolvimento deste trabalho de conclusão, relatando todos os passos feitos até chegar ao protótipo SIS (Sistema Interpretador Scout) para posteriormente analisar seus resultados e propor melhorias e ajustes para trabalhos futuros. Dessa forma, no capítulo são abordados as ferramentas utilizadas, gramática, especificação da gramática em BNF, especificação da gramática com o JavaCC, protótipo SIS, comportamento da gramática.

4.1 Ferramentas utilizadas

No desenvolvimento do protótipo SIS foram utilizados as seguintes ferramentas:

- Editor PSPad Freeware editor 4.5.5 (PSPad, 2009);
- A plataforma Java (JAVA, 2009);
- A plataforma JavaCC 4.2 (JAVA.NET, 2009);
- O ambiente de desenvolvimento NetBeans IDE 6.5.1 (NetBeans IDE, 2009).

4.2 Gramática

Como o analisador sintático é construído sobre uma gramática livre de contexto e essa gramática é composta de uma série de regras que descrevem quais são as construções válidas ou não da linguagem (DELAMARO, 2004), para o desenvolvimento do sistema interpretador scout foi necessário à construção dessa gramática através dos comandos pré-estabelecidos do sistema do Raimann (2007).

Uma linguagem consiste essencialmente de uma seqüência de *strings* ou símbolos com suas regras para definir quais seqüências de símbolos são válidas na linguagem, ou seja, a sintaxe da linguagem. A interpretação do significado de uma seqüência válida de símbolos corresponde à semântica da linguagem (IVAN, 2003).

A gramática do sistema interpretador scout foi dividida da seguinte forma: Os fundamentos do voleibol e seus respectivos atributos.

- Fundamentos de saque
    - Número da camiseta;
    - Posição do jogador;
    - Tipo de saque;
    - Direção do saque;
    - Resultado do saque.
- Fundamentos de recepção
    - Número da camiseta;
    - Posição do jogador;
    - Resultado da recepção;
    - Tipo de recepção.
- Fundamentos de levantamento
    - Número da camiseta;
    - Posição do jogador;
    - Tipo de levantamento;
    - Qualidade levantamento;
    - Direção do levantamento.
- Fundamentos de cortada
    - Número da camiseta;
    - Posição do jogador;
    - Direção da cortada;
    - Resultado da cortada;
    - Velocidade da cortada.
- Fundamentos de bloqueio

- Número da camiseta;
- Posição do jogador;
- Constituição do bloqueio;
- Resultado do bloqueio.

Devido ao fato que todos os fundamentos possuem nos seus atributos obrigatoriamente o “Número da camiseta” e a “Posição do jogador” e pelo fato dos valores dos atributos serem muitos iguais (numéricos) foi necessário criar identificadores para poder reconhecer os atributos e evitar ambigüidades na gramática. Na Figura 4.1 possui os identificadores utilizados na gramática, no qual a representação “(??)” são os valores já citados na seção 1.2.

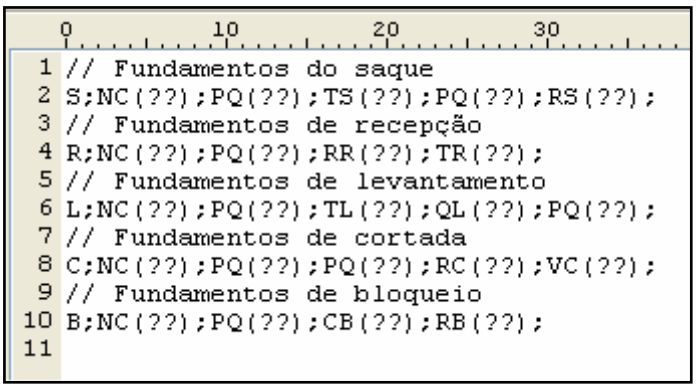

Figura 4.1- Identificadores.
Fonte: autor

A sintaxe da gramática utilizada pelo sistema de interpretação scout foi dividida em duas representações (Onde serão mais detalhados os identificadores da gramática):

- Gramática em BNF;
- Gramática em JavaCC;

Isto foi necessário, pois como JavaCC utiliza notações bem semelhantes a BNF e desta forma, ficaria bem mais fácil a implementação no JavaCC e por sua vez a criação do protótipo. Na Figura 4.2 é representado as etapas da criação da gramática até o *parser* (Sistema Interpretador Scout).

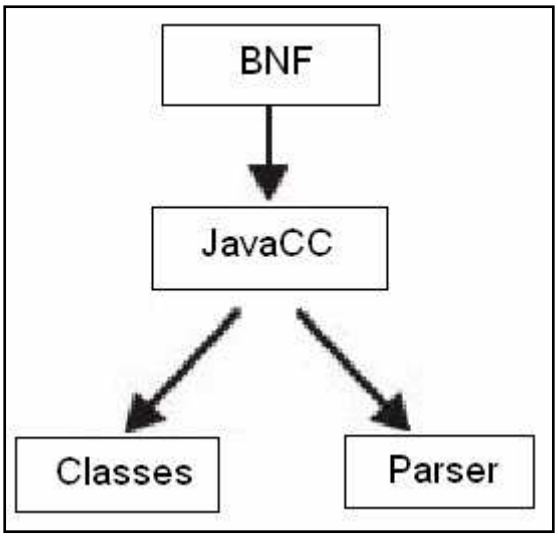

Figura 4.2- Etapas da criação da gramática.
Fonte: autor

4.2.1 Especificação da gramática em BNF

A sintaxe de uma linguagem de programação pode ser descrita por uma gramática independente de contexto e representada graficamente através de uma notação BNF (Forma de Backus-Naur). Esta gramática descreve recursivamente a combinatória de *tokens* possíveis de uma linguagem, podendo também ser usada para decompor um texto em unidades estruturais para serem organizadas dentro de um bloco. O analisador sintático recebe do analisador léxico um grupo de *tokens* e usa um conjunto de regras para construir uma árvore sintática da estrutura (LOUDEN, 2004).

Algumas características da gramática BNF (AHO, 1995):

- Ajuda a entender como se descreve programas sintaticamente correto;
- Especifica de forma inequívoca a gramática de uma linguagem;
- Pode ser usada para determinar se um programa está sintaticamente correto (Papel do compilador);
- É um conjunto de definições ou regras.

Após ter especificado as regras dos fundamentos; atributos e seus respectivos valores; foi possível desenvolver a gramática utilizando anotação em BNF.

Para maior entendimento das regras criadas em BNF, foi dividido a gramática nas seguintes partes:

- Principal: Onde foram declarados todos os fundamentos do voleibol e quais serão seus atributos. Na Figura 4.3 é demonstrado a gramática principal em BNF.

```
/*********************************************************************
    Trabalho de conclusão: Gramática de comandos Scout no voleibol
                      Diosef Da Silva Niederauer
                       Versão BNF - Scout 1.0
                             05/11/2009
*********************************************************************/
// Gramática em BNF de fundamentos do voleibol
<FUNDAMENTOS> :: =  begin <SAQUE>    end |
                    begin <RECEPCAO> end |
                    begin <LEVANTAM> end |
                    begin <CORTADA>  end |
                    begin <BLOQUEIO> end
// Fundamento de saque
   <SAQUE> :: = <NC><PQ><TS><PQ><RS>
// Fundamento de recepção
<RECEPCAO> :: = <NC><PQ><RR><TR>
// Fundamento de levantamento
<LEVANTAM> :: = <NC><PQ><TL><QL><PQ>
// Fundamento de cortada
 <CORTADA> :: = <NC><PQ><PQ><RC><VC>
// Fundamento de bloqueio
<BLOQUEIO> :: = <NC><PQ><CB><RB>
```

Figura 4.3- Gramática principal em BNF.
Fonte: autor

- *Tokens* do fundamento do saque: Possui os valores que poderão ser aceitos para o saque. Na Figura 4.4 é demonstrado os valores do fundamento de saque em BNF.

```
/*********************************************************************
//                 Tokens do fundamento de saque
/*********************************************************************
// Número da camisa
// Posição do jogador
// Tipo de saque
//(V=Viagem, F=Flutuante, P=Parado)
  <TS> :: = [V] | [F] | [P]
// Direção do saque
// Resultado do saque
//(0=Erro saque, 1=Defesa ideal, 2-Defesa não ideal, 3=Ponto)
  <RS> :: = [0] | [1] | [2] | [3]
```

Figura 4.4- *Tokens* do saque em BNF.
Fonte: autor

- *Tokens* do fundamento de recepção: Possui os valores que poderão ser aceitos para a recepção. Na Figura 4.5 é demonstrado os valores do fundamento de recepção em BNF.

```
35 /*************************************************************************
36 //                  Tokens do fundamento de recepção
37 /*************************************************************************
38 // Número da camisa
39 // Posição do jogador
40 // Resultado da recepção
41 // (0=Erro recepção, 1=Recepção não ideal, 2=Recepção ideal)
42   <RR> :: = [0] | [1] | [2]
43 // Tipo de recepção
44 // (D=Defesa, P=Passe)
45   <TR> :: = [D] | [P]
```

Figura 4.5- *Tokens* de recepção em BNF.
Fonte: autor

- *Tokens* do fundamento de levantamento: Possui os valores que poderão ser aceitos para o levantamento. Na Figura 4.6 é demonstrado os valores do fundamento de levantamento em BNF.

```
46 /*************************************************************************
47 //                  Tokens do fundamento de levantamento
48 /*************************************************************************
49 // Número da camisa
50 // Posição do jogador
51 // Tipo de levantamento
52 // (S=Suspensão, C=Chão)
53   <TL> :: = [S] | [C]
54 // Qualidade do levantamento
55 // (C=Chutada, S=Segurança, M=Meio, E=Enforcada)
56   <QL> :: = [C] | [S] | [M] | [E]
57 // Direção do levantamento
```

Figura 4.6- *Tokens* de levantamento em BNF.
Fonte: autor

- *Tokens* do fundamento de cortada: Possui os valores que poderão ser aceitos para a cortada. Na Figura 4.7 é demonstrado os valores do fundamento de cortada em BNF.

```
58 /*********************************************************************
59 //                    Tokens do fundamento de cortada
60 /*********************************************************************
61 // Número da camisa
62 // Posição do jogador
63 // Direção da cortada
64 // Resultado da cortada
65 // (0=Erro cortada, 1=Defesa ideal, 2=Defesa não ideal,3=Ponto atacante)
66   <RC> :: = [0] | [1] | [2] | [3]
67 // Velocidade da cortada
68 // (R=Rápida, P=Pingada)
69   <VC> :: [R] | [P]
```

Figura 4.7- *Tokens* de cortada em BNF.
Fonte: autor

- *Tokens* do fundamento de bloqueio: Possui os valores que poderão ser aceitos para o bloqueio. Na Figura 4.8 é demonstrado os valores do fundamento de bloqueio em BNF.

```
70 /*********************************************************************
71 //                  Tokens do fundamento de bloqueio
72 /*********************************************************************
73 // Número da camisa
74 // Posição do jogador
75 // Constituição do bloqueio
76 // (T=Triplo, D=Duplo, S=Simples)
77   <CB> :: = [T] | [D] | [S]
78 // Resultado do bloqueio
79 // (0=Erro bloqueio, 1=Houve bloqueio retornou, 2=Houve bloqueio ficou,
80 //  3=Ponto bloqueio)
81   <RB> :: = [0] | [1] | [2] | [3]
```

Figura 4.8- *Tokens* de bloqueio em BNF.
Fonte: autor

- *Tokens* universais: Como possui atributos que serão aceitos em todos os fundamentos (Número da camiseta e Posição do jogador) foi criado os tokens universais, para não ser preciso declará-los em todos os fundamentos. Na Figura 4.9 é demonstrado os valores dos tokens universais em BNF.

```
82 /*********************************************************************
83 //       Tokens universais (Utilizados por todos os fundamentos)
84 /*********************************************************************
85 // Número da camisa
86   <NC> :: = [0-9]
87 // Posição na quadra de volei
88   <PQ> :: = [1] | [2] | [3] | [4] | [5] | [6]
89 /*********************************************************************
```

Figura 4.9- *Tokens* universais em BNF.
Fonte: autor

4.2.2 Especificação da gramática com o JavaCC

Após as definições da gramática em BNF, estas definições foram convertidas para o padrão do JavaCC.

O JavaCC toma como entrada uma gramática e transforma-a num programa descrito em linguagem Java capaz de analisar um arquivo e dizer se satisfaz ou não as regras especificadas nesta gramática. Oferece facilidades para a construção da árvore sintática. O programa gerado pelo JavaCC realiza um tipo de análise sintática chamada de top down ou análise descendente (DELAMARO, 2004).

A criação de um interpretador utilizando o JavaCC começa pela definição da gramática, normalmente em um arquivo com extensão "jj", apesar de não obrigatório. De um modo geral, um arquivo do JavaCC segue uma sintaxe simples, sendo formado pela definição das opções de geração, seguido pela especificação do *parser* e produções da gramática (DELAMARO, 2004). Na Figura 4.10 é representado as entradas e saídas do JavaCC com a definição do *parser* como Scout.jj.

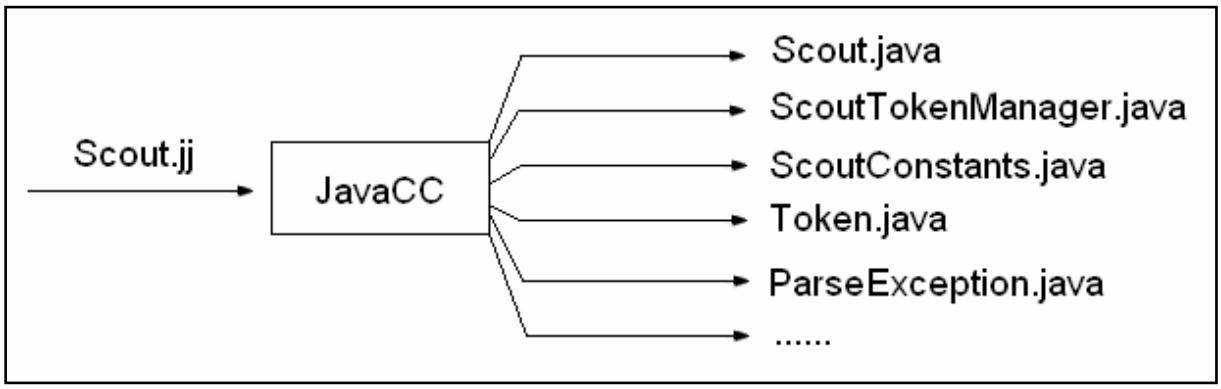

Figura 4.10- Entradas e saídas do JavaCC.
Fonte: autor

Basicamente as etapas necessárias para a criação de uma gramática utilizando o JavaCC, são as seguintes (DELAMARO, 2004):

- Lista de opções (Opcional) – O JavaCC possui parâmetros que podem ser especificados tanto no arquivo da gramática quanto na linha de comandos. Alguns exemplos: “LOOKAHEAD” (Esta opção indica quantos *tokens* (palavra) o analisador deve verificar antes de tomar uma decisão. Quanto menor o valor de *lookahead*, mais rápido será a análise. Seu default é um.); “STATIC” (Esta é uma opção *booleana* em que o seu valor default é verdadeiro (true). Se declarada como verdadeiro, todos os métodos e

variáveis da classe serão especificadas como estáticos no *parser* gerado. Isto permite só um objeto *parser* pode estar presente, mas isto melhora o desempenho do *parser*.), etc. Para maiores detalhes sugere-se a leitura do *help* do JavaCC. Na Figura 4.11, é representada um modelo para a definição das opções.

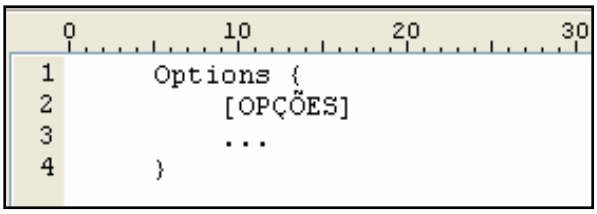

Figura 4.11- Lista de opções.
Fonte: autor

- Unidade de compilação JavaCC – Aqui é definido o nome da classe que irá abrigar o analisador sintático. Tudo que aparece entre esses dois comandos será inserido como código nesta classe. Assim, podem ser inseridos *imports*, variáveis, métodos etc. Na Figura 4.12, é representada um modelo para a definição da unidade de compilação.

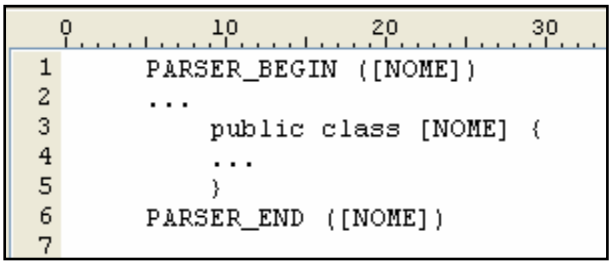

Figura 4.12 - Estrutura.
Fonte: autor

- Lista de produções da gramática – O resto das declarações são as produções da gramática que se deseja implementar, onde o JavaCC utiliza a técnica descendente recursiva de análise. Na Figura 4.13, é representado um modelo para a lista de produções da gramática.

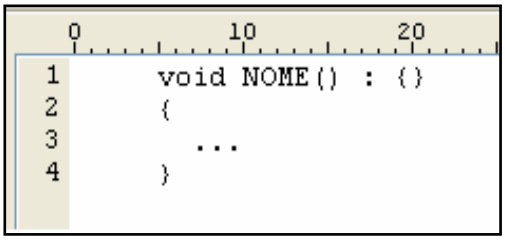

Figura 4.13 – Produção.
Fonte: autor

O JavaCC possui também algumas especificações léxicas como por exemplo: *SKIP* (Ignora *token* definidos neste região) e *TOKEN* (Utilizado para especificar o padrão de um *token*).

O JavaCC permite que as produções sejam colocadas entre construções *try/catch* e caso algum erro sintático seja detectado ao tentar casar a entrada com essas produções, o erro pode ser tratado.

Após analisar os principais aspectos para a utilização do JavaCC e com a gramática finalizada em BNF, foi possível desenvolver a gramática utilizando anotações no JavaCC. Para maior entendimento das regras criadas no JavaCC, foi dividido a gramática da seguinte forma:

- Principal: Onde é declarado a unidade de compilação, indicando inicio e fim do *parser*. Neste bloco é chamado as regras dos fundamentos do voleibol, estando preparado tanto para ser chamado diretamente do *Prompt* (Através de linha de comando: Java Scout) como do NetBeans (Será visto mais adiante no próximo capitulo). Na Figura 4.14 é demonstrado a gramática principal em JavaCC.

```
/*************************************************************************
    Trabalho de conclusão: Gramática de comandos Scout no voleibol
                      Diosef Da Silva Niederauer
                       Versão JavaCC - Scout 1.0
                             05/11/2009
*************************************************************************/
// Inicio do parser
PARSER_BEGIN(Scout)
// Funcionalidades básicas para filtragem de saída de dados
// Arranjo de bytes
import java.io.ByteArrayInputStream;
        public class Scout {
        public static void main(String args[]) throws ParseException {
// Se não foi passado como parametro a entrada (Prompt)
        if (args.length == 0) {
           Scout parser = new Scout(System.in);
// Exibe no Prompt a versão do Scout
           {System.out.println("");}
           {System.out.println("Versão Scout 1.0");}
           {System.out.println("");}
// Regras dos fundamentos
           Scout.Fundamentos();
        }
// Se foi passado como parametro a entrada (NetBeans)
        else
        {
           ByteArrayInputStream comandos = new ByteArrayInputStream(args[0].getBytes());
           Scout parser = new Scout(comandos);
// Regras dos fundamentos
           Scout.Fundamentos();
        }
}
}
// Final do parser
PARSER_END(Scout)
```

Figura 4.14 – Gramática principal em JavaCC.
Fonte: autor

- Léxico: Onde são declarados os tokens a serem ignorados (*SKIP*) e os próprios *tokens* utilizados na gramática. Na Figura 4.15 é representado os comandos ignorados. Na seqüência possui cada *token* de cada fundamento passo a passo.

```
36 // ----------------------------------------------------------
37 // Lexico
38 // ----------------------------------------------------------
39 // Símbolos que não devem ser considerados na análise
40 SKIP :
41 {
42   " "
43 | "\t"
44 | "\n"
45 | "\r"
46 | "\r\n"
47 | ";"
48 }
```

Figura 4.15 – Léxico JavaCC.
Fonte: autor

- *Tokens* Saque: Possui a letra "S" para identificar que é um fundamento do tipo saque; os caracteres "TS" para identificar que é um tipo de saque (Valores: "V", "F", "P"); os caracteres "RS" para identificar que é o resultado do saque (Valores: "0", "1", "2", "3").
- *Tokens* Recepção: Possui a letra "R" para identificar que é um fundamento do tipo recepção; os caracteres "RR" para identificar que é um resultado da recepção (Valores: "0", "1", "2"); os caracteres "TR" para identificar que é um tipo de recepção (Valores: "D", "P").
- *Tokens* Levantamento: Possui a letra "L" para identificar que é um fundamento do tipo levantamento; os caracteres "TL" para identificar que é um tipo de levantamento (Valores: "S", "C"); os caracteres "QL" para identificar a qualidade do levantamento (Valores: "C", "S", "M", "E").
- *Tokens* Cortada: Possui a letra "C" para identificar que é um fundamento do tipo cortada; os caracteres "RC" para identificar o resultado da cortada (Valores: "1", "2", "3"); os caracteres "VC" para identificar a velocidade da cortada (Valores: "R", "P").
- *Tokens* Bloqueio: Possui a letra "B" para identificar que é um fundamento do tipo bloqueio; os caracteres "CB" para identificar a constituição do bloqueio (Valores: "T", "D", "S"); os caracteres "RB" para identificar o resultado do bloqueio (Valores: "0", "1", "2", "3").

- *Tokens* Universais: São *tokens* que podem ser utilizados por todos os fundamentos. Na Figura 4.16 é representado esses *tokens*.

```
155 // Sem fundamentos informados e nem sinal de finalização
156 TOKEN :
157 {
158   < SEMFUN : ~["S","R","L","C","B","@"] >
159 }
160 // Simbolo para indicar que finalisou a rotina
161 TOKEN :
162 {
163   < EOL : "@" >
164 }
165 // Tokens utilizados por todos os fundamentos
166 // Número da camisa
167 TOKEN :
168 {
169   < NC  : <INC> (<VNC>) *>
170 | <#INC : ("NC")>
171 | <#VNC : (["0" - "9"])>
172 }
173 // Posição na quadra de volei
174 TOKEN :
175 {
176   < PQ  : <IPQ> (<VPQ>) *>
177 | <#IPQ : ("PQ")>
178 | <#VPQ : (["1","2","3","4","5","6"])>
179 }
```

Figura 4.16 – *Tokens* universais em JavaCC.
Fonte: autor

- Sintático: Onde é declarado a lista de produções da gramática, dos fundamentos do voleibol. É verificado qual o fundamento foi acionado para que seja feito sua validação, caso não seja identificado nenhum fundamento é exibido uma lista com os valores dos fundamentos validos (*try/catch*). Na Figura 4.17 possui a classe completa informada. Na seqüência é representado cada produção separadamente passo a passo.

```
0         10        20        30        40        50        60        70
181 // Sintatico
182 // ----------------------------------------------------------
183 // Fundamentos do voleibol
184 void Fundamentos() : { }
185 {
186 try
187   {
188           <SAQUE>    {} FUN_SAQ() Fundamentos()
189 |         <RECEPCAO> {} FUN_REC() Fundamentos()
190 |         <LEVANTAM> {} FUN_LEV() Fundamentos()
191 |         <CORTADA>  {} FUN_COR() Fundamentos()
192 |         <BLOQUEIO> {} FUN_BLO() Fundamentos()
193 |                             ((<EOF> | <EOL>))
194   }
195 catch (ParseException e)
196   {
197    (System.out.println("");
198     System.out.println("##############################################");
199     System.out.println("###     NÃO É UM FUNDAMENTO VÁLIDO      ###");
200     System.out.println("##############################################");
201     System.out.println("###              UTILIZAR                  ###");
202     System.out.println("### S - Para fundamento de saque         ###");
203     System.out.println("### R - Para fundamento de recepção      ###");
204     System.out.println("### L - Para fundamento de levantamento ###");
205     System.out.println("### C - Para fundamento de cortada       ###");
206     System.out.println("### B - Para fundamento de bloqueio      ###");
207     System.out.println("##############################################");}
208   }
209 }
```

Figura 4.17 – Sintático JavaCC.
Fonte: autor

- Sintático Saque: A seqüência válida é Número da camiseta (NC); Posição do jogador (PQ); Tipo de saque (TS); Direção do saque (PQ); Resultado do saque (RS) e caso esta seqüência não seja completa o sistema vai exibir a seguinte mensagem "*** Não foram informados todos os atributos do saque ***", porém os atributos que ele reconhecer ele listará.

- Sintático Recepção: A seqüência válida é Número da camiseta (NC); Posição do jogador (PG); Resultado da recepção (RR); Tipo de recepção (TR) e caso esta seqüência não seja completa o sistema vai exibir a seguinte mensagem "*** Não foram informados todos os atributos da recepção ***", porém os atributos que ele reconhecer ele listará.

- Sintático Levantamento: A seqüência válida é Número da camiseta (NC); Posição do jogador (PG); Tipo de levantamento (TL); Qualidade do levantamento (QL); Direção do levantamento (PQ) e caso esta seqüência não seja completa o sistema vai exibir a seguinte mensagem "*** Não foram informados todos os atributos do levantamento ***", porém os atributos que ele reconhecer ele listará.
- Sintático Cortada: A seqüência válida é Número da camiseta (NC); Posição do jogador (PG); Direção da cortada (PQ); Resultado da cortada (RC); Velocidade da cortada (VC) e caso esta seqüência não seja completa o sistema vai exibir a seguinte mensagem "*** Não foram informados todos os atributos da cortada ***", porém os atributos que ele reconhecer ele listará.
- Sintático Bloqueio: A seqüência válida é Número da camiseta (NC); Posição do jogador (PG); Constituição do bloqueio (CB); Resultado do bloqueio (RB) e caso esta seqüência não seja completa o sistema vai exibir a seguinte mensagem "*** Não foram informados todos os atributos do bloqueio ***", porém os atributos que ele reconhecer ele listará.

Após a construção da gramática em JavaCC é necessário compilar a gramática. No *Prompt* onde se encontra o programa do JavaCC (mais precisamente no diretório "bin") deve rodar o comando "**JavaCC Scout.jj"** (Verificar que o sistema ira gerar os arquivos na extensão *.java), logo após é necessário compilar os códigos javas através do comando "**javac *.java -Xlint:unchecked".** Desta forma esta pronto para utilizar o analisador sintático rodando **java Scout,** na Figura 4.18 possui um exemplo simples com o fundamento de bloqueio com todos seus atributos rodando diretamente do analisador (o símbolo "@" significa o final da gramática).

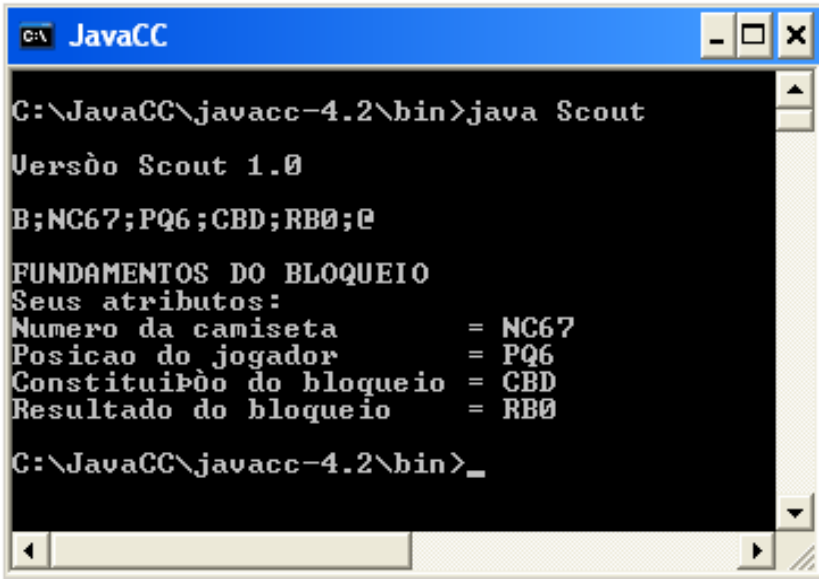

Figura 4.18 – Java Scout.
Fonte: autor

4.3 Protótipo SIS

NetBeans IDE é um aplicativo de código aberto utilizando a tecnologia Java web e celulares. Esta ferramenta cria design e componentes para interfaces. Também é indicada para eliminar erros em desenvolvimentos para celulares. O aplicativo permite a modelagem de visuais UML através de diversos diagramas desenhando e analisando as aplicações do modelo. NetBeans IDE é indicado tanto para profissionais como para amadores (NetBeans, 2009).

Devidos a grande facilidade de utilizar esta ferramenta à mesma foi escolhida para ser desenvolvido a interface que ira receber os comandos de fundamentos *scout* de voleibol. Na Figura 4.19 é demonstrado a janela de desenvolvimento do NetBean, possuindo recursos bem fáceis de ser utilizados (com pouca prática já é possível criar interfaces).

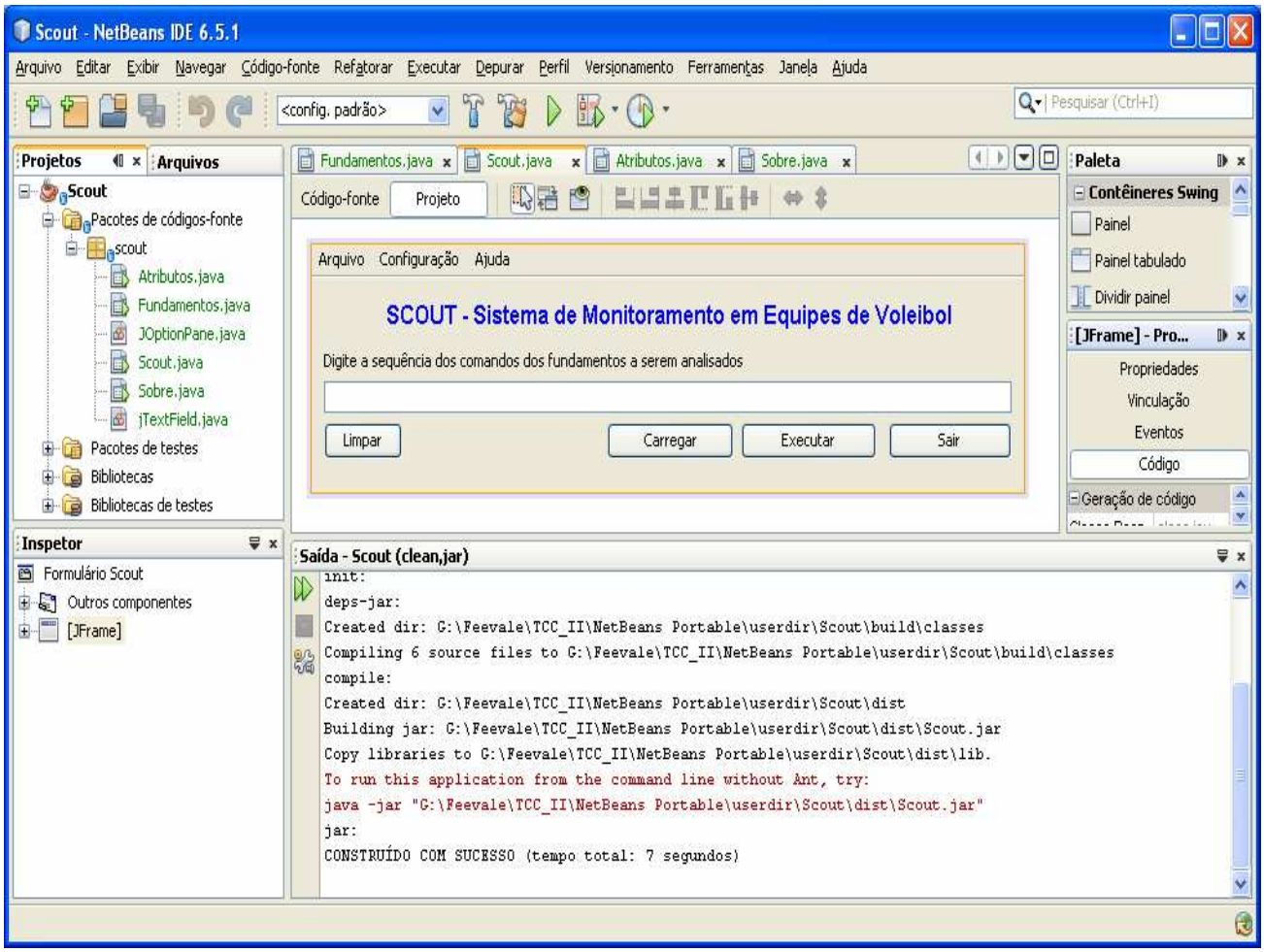

Figura 4.19 – NetBeans.
Fonte: autor

Basicamente o protótipo foi construído da forma a ser mais simples possível, porém possuindo todos os recursos necessários para poder configurar os fundamento e seus atributos.

Outro recurso que o sistema ira disponibilizar é poder efetuar testes da gramática sem fazer a integração com o sistema Raimann (2007), gerando por si próprio um relatório com os fundamentos e seus atributos reconhecidos.

A interface do Sistema Interpretador Scout ficou bem compacta, possuindo na janela principal um campo para ser informado a seqüência dos comandos dos fundamentos a serem analisados; recurso para inicializar o campo digitado; recurso para carregar através de arquivos os fundamentos a serem analisados, recurso para executar diretamente do campo e algumas configuração como: Valores de fundamentos e Valores de atributos. Na Figura 4.20 é apresentado a janela principal do SIS.

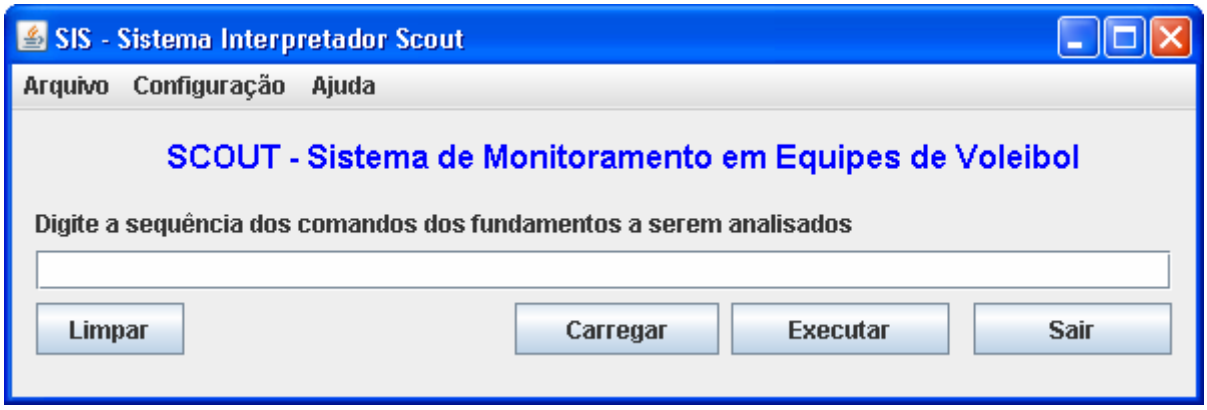

Figura 4.20 – SIS – Sistema Interpretador Scout.
Fonte: autor

Acessando as configurações possui a opção de “Fundamento do voleibol”, onde é possível alterar os valores padrões para os fundamentos. Caso venha a precisar, possui o recurso de poder carregar novamente os padrões através do botão “Padrão”. Por enquanto não está salvando estes campos, somente em tempo de execução do sistema. Na Figura 4.21 possui a janela dos fundamentos do voleibol.

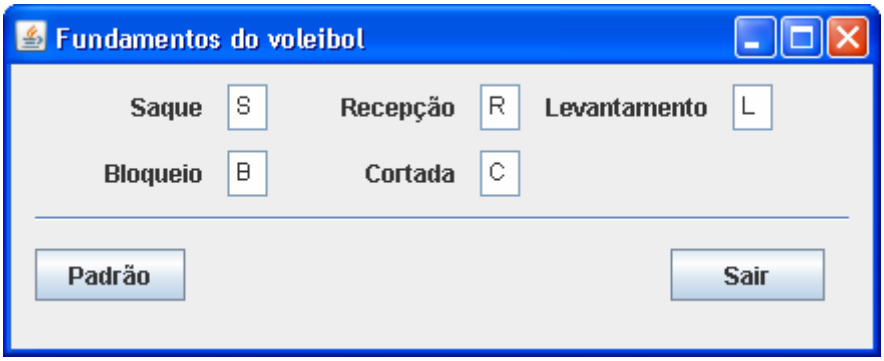

Figura 4.21 – SIS – Fundamentos do voleibol.
Fonte: autor

Possui também nas configurações a opção de “Atributos do voleibol”, onde é possível alterar os valores padrões para os atributos de todos os fundamentos. Caso venha a precisar, possui o recurso de poder carregar novamente os padrões através do botão “Padrão”. Da mesma forma que na opção de "Fundamentos do voleibol" não está salvando estes campos, somente em tempo de execução do sistema. Na Figura 4.22 possui a janela dos atributos do voleibol.

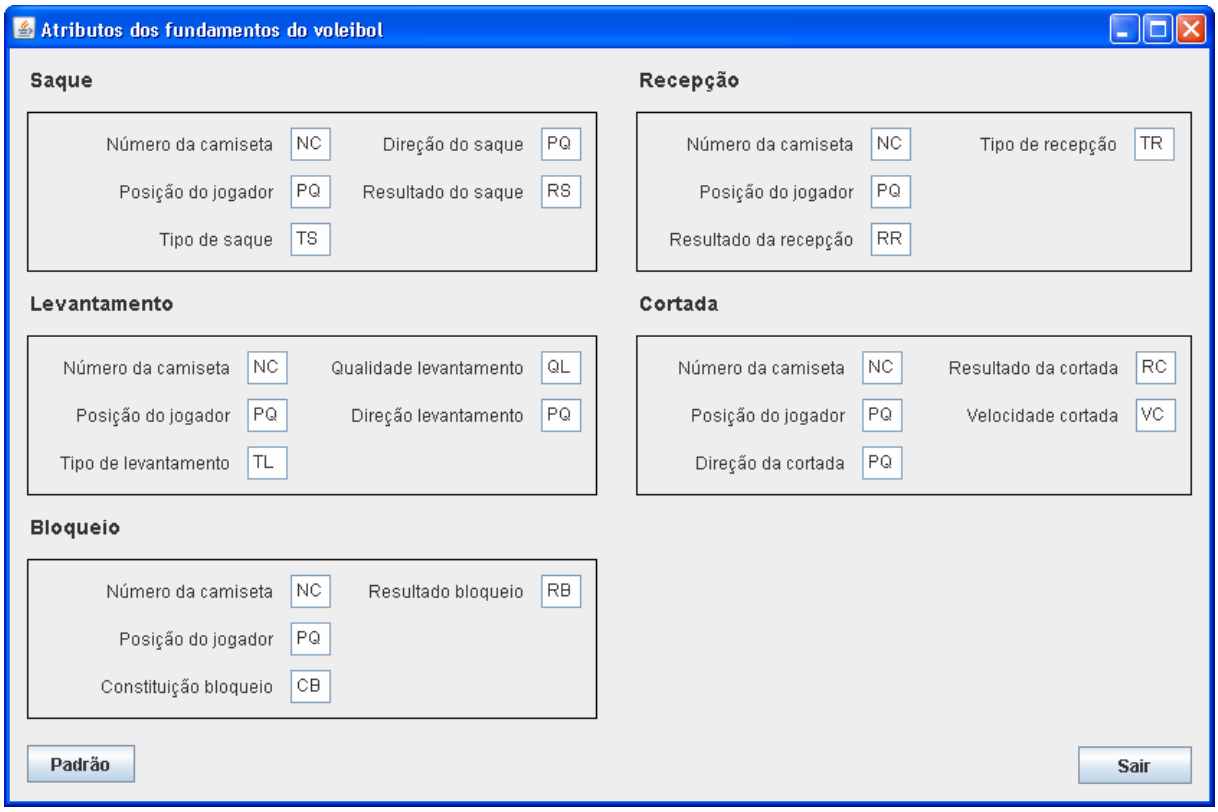

Figura 4.22 – SIS – Atributos dos fundamentos do voleibol.
Fonte: autor

O protótipo possui algumas validações básicas, tais como:

- Exibir mensagens informando que não foi possui nenhuma seqüência para o sistema poder executar;
- Não fazer nada ao clicar em carregar caso não tenha selecionando nenhum arquivo;
- Não rodar ao clicar em executar caso não tenha informado nenhum arquivo de saída.

4.4 Comportamento da gramática

Terminado a construção do Sistema Interpretador Scout, foi necessário submeter o sistema numa bateria de testes para viabilizar suas usabilidades e praticidade perante o *Scouter*. O sistema deve prever que o usuário não informou todos os fundamentos com seus respectivos atributos para poder registrar somente os fundamentos válidos e também pode (se for o caso) parametrizar o sistema da forma que mais lhe agrada, tornando o sistema bem completo.

Serão relatados cinco testes feitos no Sistema Interpretador Scout, onde:

- Primeiro teste: O SIS foi carregado com um arquivo chamado “Teste1.txt” que contêm os cinco fundamentos de voleibol com todos seus atributos informados (todos separados por “;”), após ser carregado pelo sistema e ter sido informado o nome de saída como “Resultado1.txt”, teremos o seguinte resultado conforme a Figura 4.23.

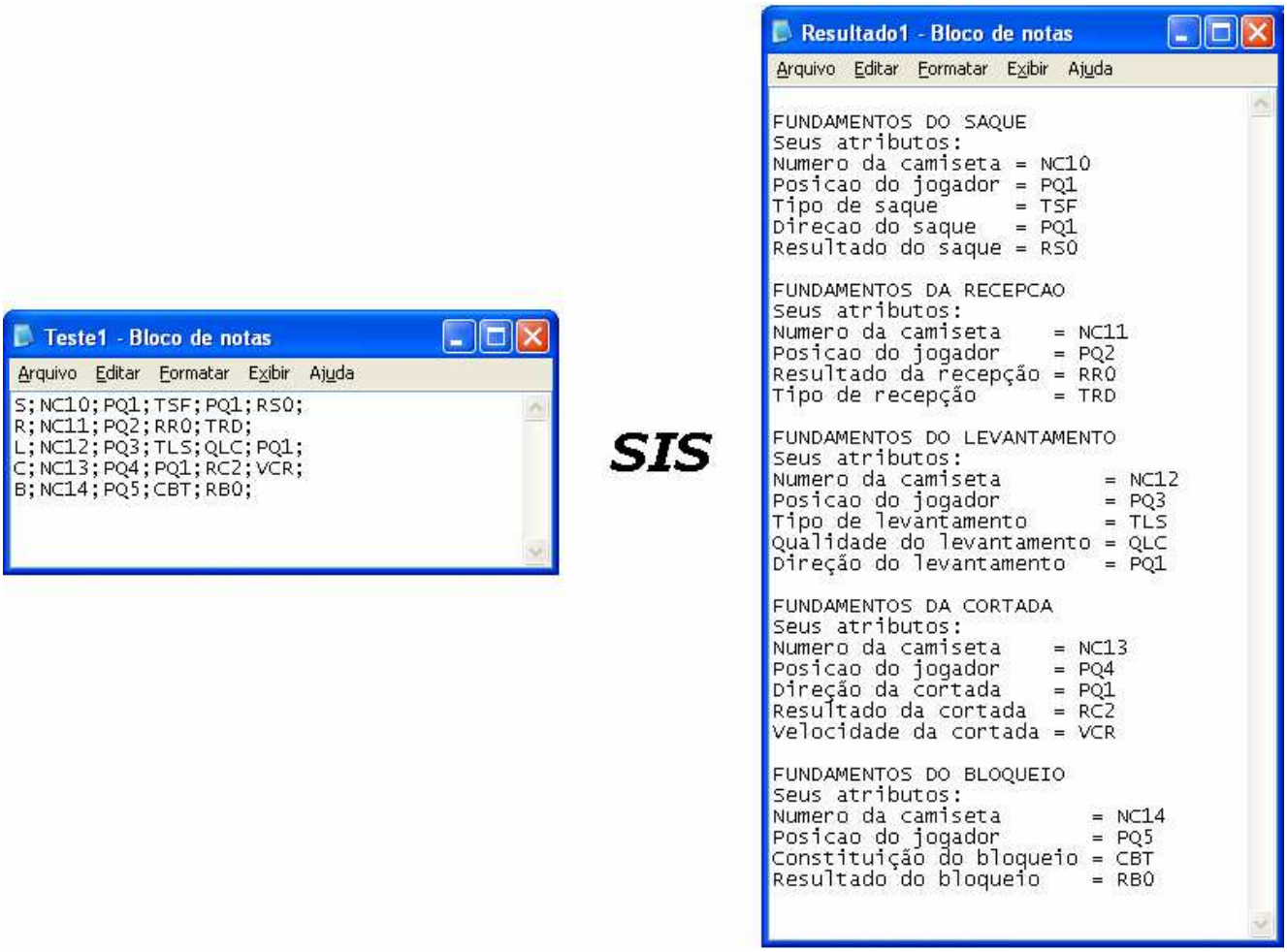

Figura 4.23 – SIS – Teste1.
Fonte: autor

- Segundo teste: O SIS foi carregado com um arquivo chamado “Teste2.txt” que contêm três fundamentos de saque com todos seus atributos (porém valores diferentes), um fundamento de bloqueio sem nenhum atributo, um fundamento de cortada sem nenhum atributo, um fundamento de levantamento sem nenhum atributo e um fundamento de recepção sem nenhum atributo (todos separados por “;”), após ser carregado pelo sistema e

ter sido informado o nome de saída como “Resultado2.txt”, teremos o seguinte resultado conforme a Figura 4.24.

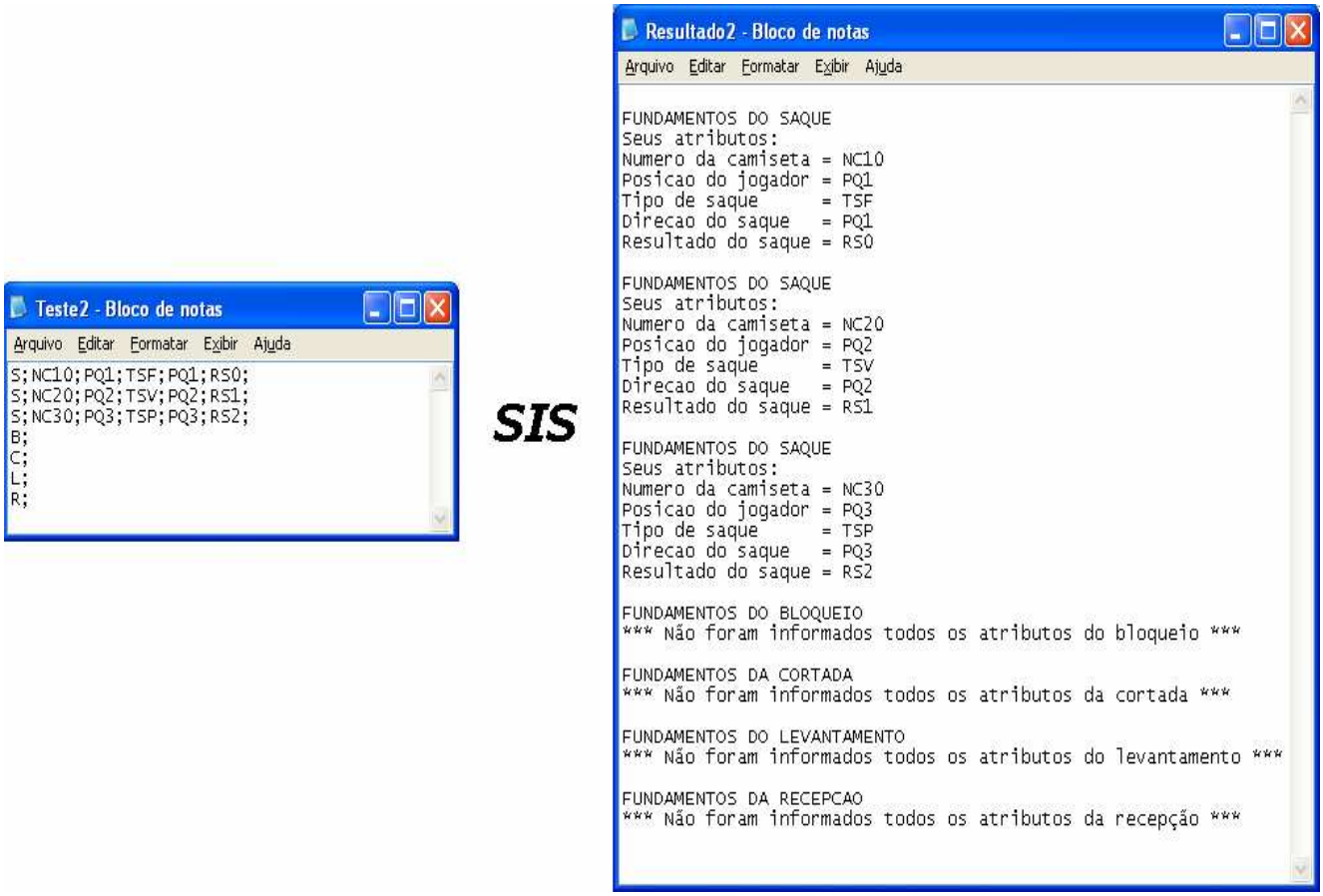

Figura 4.24 – SIS – Teste2.
Fonte: autor

- Terceiro teste: O SIS foi carregado com um arquivo chamado “Teste3.txt” que contêm somente os fundamentos sem nenhum atributo informado, após ser carregado pelo sistema e ter sido informado o nome de saída como “Resultado3.txt”, teremos o seguinte resultado conforme a Figura 4.25.

Figura 4.25 – SIS – Teste3.
Fonte: autor

Quarto teste: O SIS foi carregado com um arquivo chamado "Teste4.txt", porém antes foi configurado os fundamentos e atributos do SIS, pois o arquivo possui valores diferenciados do padrão. Na configuração do fundamento foi configurado no campo de "Saque" com o caractere "Z" e para a configuração dos atributos relativos ao "Saque" foi configurado da seguinte forma: Número da camiseta = "AA"; Posição do jogador = "BB"; Tipo de saque = "CC"; Direção do saque = "DD" e Resultado do saque = "EE". Após ser carregado pelo sistema e ter sido informado o nome de saída como "Resultado4.txt", temos o seguinte resultado conforme a Figura 4.26

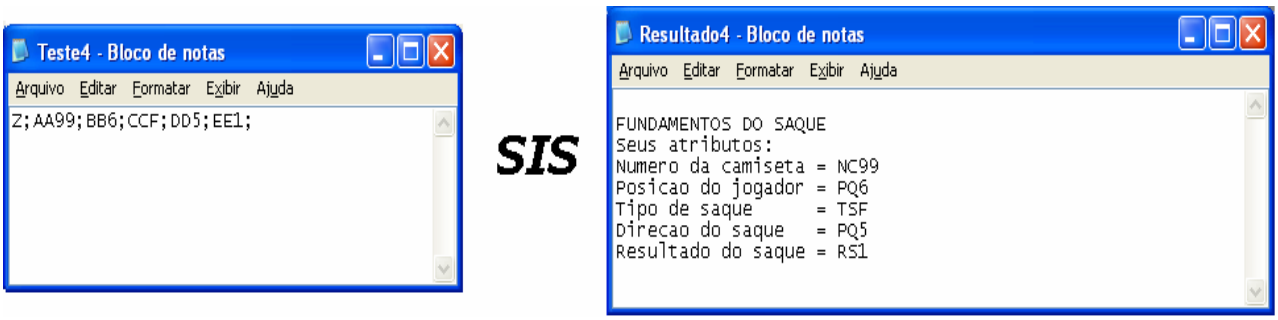

Figura 4.26 – SIS – Teste4.
Fonte: autor

- Quinto teste: O SIS foi carregado com um arquivo chamado "Teste5.txt" que contêm todos os fundamentos, porém sem seus respectivos atributos e um fundamento invalidado para testar se o sistema ira avisar que o fundamento

informado não é um valido. Após ser carregado pelo sistema e ter sido informado o nome de saída como “Resultado5.txt”, teremos o seguinte resultado conforme a Figura 4.27.

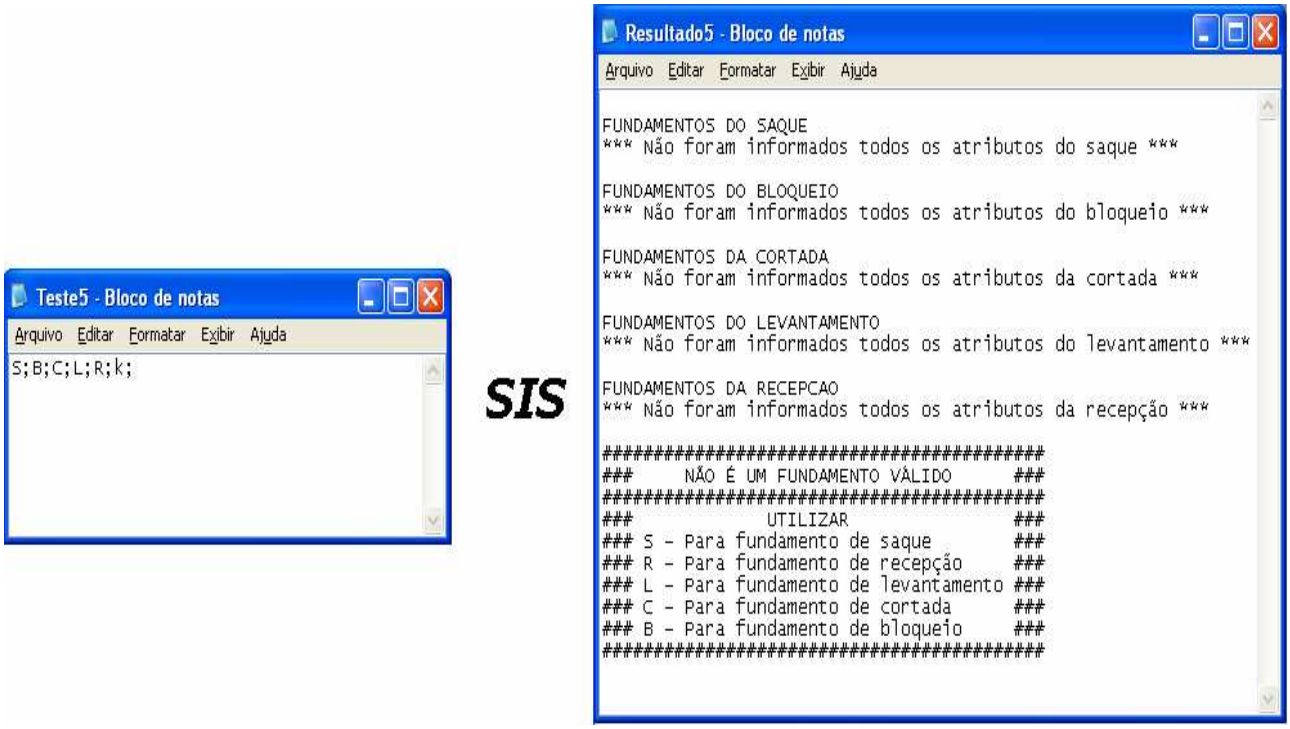

Figura 4.27 – SIS – Teste5.
Fonte: autor

Os testes demonstrados neste trabalho de conclusão foram todos recebendo como arquivos os parâmetros das seqüências dos fundamentos do voleibol, porém se forem feitos diretamente no Sistema Interpretador Scout (campo de digitação) o resultado é o mesmo.

CONCLUSÃO

Através dos estudos realizados junto ao projeto de pesquisa "A IA entrando na quadra de vôlei: *scout* inteligente" verificou-se a necessidade de ser modificada a gramática de comandos *scout* do sistema desenvolvido por Raimann (2007), pois para registrar todos os eventos ligados aos fundamentos de um ponto de uma partida, dependendo da agilidade da pessoa com o teclado, pode ser extremamente lento, o que acaba inviabilizando totalmente o uso desse tipo de recurso.

Foram levantados também a necessidade de criar uma interface diferenciada para se poder registrar a entrada de fundamentos, facilitando a forma de parametrizar quais fundamentos analisar, ou seja, quais teclas ou símbolos usar para o registro dos fundamentos.

Em função desses fatores, surgiu a proposta de analisar, e propor um protótipo de um interpretador de comandos scout do voleibol. Seguindo as mesmas características de um software livre e voltado para web, da mesma forma que o software do Raimann (2007) propôs, podendo assim, ser utilizado sem restrições por qualquer interessado no assunto.

Infelizmente, para uma definição mais precisa das ações realizadas por um *scouter* é necessário dominar o processo de cadastro de comandos de softwares proprietários, como o Data Volley. Entretanto, esses softwares são extremantes restritos a grandes equipes de voleibol, além de seus valores de aquisição, o que acaba prejudicando a elaboração deste projeto de conclusão de curso.

Durante a elaboração do presente trabalho, também foram encontradas algumas dificuldades em utilizar a ferramenta JavaCC, pois os exemplos encontrados ou eram muito simples, ou eram muitos complexos, e pelo fato do autor ter pouco conhecimento em Java acabou agravando a situação.

O trabalho apresenta um aprofundamento no estudo bibliográfico sobre Compiladores, Interpretadores. Apresenta também o estudo dos sistemas de *Scout* existentes

no mercado e principalmente o sistema desenvolvido pelo Raimann (2007) que serviu de base para este trabalho de conclusão. Para o desenvolvimento do projeto, no trabalho realizou-se o estudo, a análise e a apresentação: i) gramática em BNF; ii) gramática em JavaCC; iii) das telas do protótipo; iv) testes elaborados para validação do projeto.

Através do testes elaborados foi possível identificar a facilidade de uso, otimização e flexibilidade da digitação de comandos scout durante uma partida de voleibol na utilização do SIS (Sistema Interpretador Scout), viabilizando o protótipo.

Como trabalhos futuros, sugere-se:

- Gravar os campos configuráveis do Sistema Interpretador Scout (Fundamentos, Atributos);
- Implementar a integração do interpretador com o sistema de scout de Raimann (2007);
- Apresentação do protótipo a profissionais de educação física e avaliação de resultados.

REFERÊNCIAS BIBLIOGRÁFICAS

AHO, Alfred. **Compiladores**. Princípios, Técnicas e Ferramentas. Rio de Janeiro: LTC, 1995.

A. M. A. Price, S. S. Toscani. **Implementação de Linguagens de Programação**: Compiladores. 3ª ed. Porto Alegre: Sagra-Luzzatto. 2005. Cap 3, até Seção 3.2.2.

BALIEIRO, S. **Jogada de alta tecnologia**. INFO: tecnologia da informação, número 224, ano 19, novembro 2004.

BERNARDINHO. **Transformando suor em ouro**. Rio de Janeiro, RJ: Sextante, 2006. 215 p.

BINDER, Fábio Vinícius. **Sistemas de apoio à decisão**. São Paulo, SP: Érica, 1994.98 p.

BUTZEN, Émerson, **Proposta de um módulo de Data Mining para sistema de Scout no voleibol**. Novo Hamburgo, RS: 2008. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Instituto de Ciências Exatas e Tecnológicas, Feevale, 2008.

DATAPROJECT. Download do **Data Volley 2007 Lite Version**. Disponível em: <http://www.dataproject.com>. Acesso em 01/09/2008.

DEITEL, H. M. and Deitel, P. J. (2001). **Java como programar**. Editora: Bookman.

DELAMARO, M.E. **Como construir um compilador utilizando ferramentas Java**. Ed.Novatec, São Paulo. 2004.

FIVB, **Fédération Internationale de Volleyball**. Disponível em: <http://http://www.fivb.org/>. Acesso em: 11 de junho 2009.

GUIMARÃES, José de Oliveira (2007). **Construção de Compiladores**. Departamento de Computação – UFSCar – São Carlos, SP.

IVAN L. M. Ricarte, **Introdução à Compilação.** São Paulo, Ed. Elsevier, 2003.

JAVA. **Java Home.** Disponível em: <http://java.sun.com/javase/>. Acesso em: 15 de junho 2009.

JAVA.NET. CC, **The source for Java Technology Collaboration**. Disponível em: <https://javacc.dev.java.net/>. Acesso em: 22 de junho 2009.

João, P. (2004). **Efeitos da qualidade da recepção do serviço na efectividade do ataque**. Estudo comparativo da prestação dos jogadores líbero e recebedores prioritários em equipas de elevado rendimento competitivo no voleibol. Tese apresentada às provas de Mestrado de Alto Rendimento no ramo de Ciência do Desporto. FCDEF-UP.

JUNQUEIRA, L. A. C. **Negociação**: tecnologia e comportamento. Rio de Janeiro: COP Editora. 1998.

LOUDEN, K. C. (2004). **Compiladores**: Princípios e Práticas. Editora: Thomson Learning.

MENEGOTTO, Vanessa. **Sistema de posicionamento em quadra de voleibol com Agentspeak(L) e Jason**. Novo Hamburgo, RS: 2008. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Instituto de Ciências Exatas e Tecnológicas, Feevale, 2008.

NetBeans IDE - **Home** – Disponível em <http://www.netbeans.org/> (Versão 6.5.1). Acesso em 02 de novembro de 2009.

PSPad. **A freeware code editor**. Disponível em: <http://www.pspad.com/>. Acesso em: 01 de maio de 2009.

RAIMANN, Luís Henrique, Scout: **Sistema de monitoração em equipes de voleibol**. Novo Hamburgo, RS: 2007. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Instituto de Ciências Exatas e Tecnológicas, Feevale,2007.

RANGEL, J. L. M. **Compiladores**. Disponível em: <http://www-di.inf.pucrio.br/~rangel/comp.html>. Acesso em: 23 de março 2009.

SCHNEIDER, C.; PASSERINO, L. M.; OLIVEIRA, R. F. **Compilador Educativo Verto**: ambiente para aprendizagem de compiladores. RENOTE - Revista Novas Tecnologias na Educação, Porto Alegre, v. 3, n. 2, 2005.

ZAMBERLAM, Alexandre de Oliveira; WIVES, Leandro Krug; GOULART, Rodrigo Rafael Villarreal; SILVEIRA, Roni Gilberto. **A IA entrando na quadra de vôlei**: scout inteligente. Hífen, Uruguaiana, RS, v.29, n.55/56, p.103-110, I/II semestre 2005.