Instituto de Ciências Matemáticas e de Computação – ICMC
Universidade de São Paulo – USP

Projeto e Implementação do Sistema SciPo

Lucas Antiqueira
Valéria Delisandra Feltrim
Maria das Graças Volpe Nunes

Relatórios Técnicos do ICMC
Nº 223

Dezembro, 2003

Resumo

É exposta aqui uma implementação do sistema SciPo, um sistema de auxílio à escrita de resumos acadêmicos em português, que está sendo construído como uma aplicação Web, através, principalmente, das linguagens PHP e Prolog. Em decorrência desse trabalho, um protótipo inicial está disponível para futuras modificações e ampliações.

Índice

1. Introdução

Neste relatório é apresentada uma implementação inicial do sistema SciPo [1], um sistema de auxílio à escrita, nesta primeira fase, de resumos acadêmicos em português, em especial de teses e dissertações em Ciências de Computação. O projeto SciPo tem sido inspirado no ambiente AMADEUS [2], um ambiente de auxílio e ensino da escrita técnica em inglês. O AMADEUS utiliza a abordagem de reutilização de expressões padrões utilizadas em textos reconhecidos como bons para que, no caso de dúvida no momento da escrita, o escritor tenha uma base de bons exemplos a qual ele possa recorrer. Esse ambiente utiliza o Raciocínio Baseado em Casos para a recuperação de exemplos e também a abordagem de críticas para a avaliação das estruturas construídas pelo usuário.

Um trabalho científico pode enquadrar-se, em geral, dentro de um esquema que já se tornou clássico pela simplicidade, pelo desenvolvimento metódico e por abranger aspectos essenciais de uma comunicação científica desse gênero. Essa estrutura esquemática pode ser enunciada como Introdução – Desenvolvimento – Conclusão, sendo que o Desenvolvimento pode desdobrar-se nas seções de Materiais e Métodos e Resultados, ou ainda Materiais e Métodos, Resultados e Discussão. Em linhas gerais, essa estrutura deve guiar o leitor e fazer com que ele siga o movimento geral-para-específico, realizado na Introdução, e específico-para-geral, realizado na Conclusão. O Resumo é um componente independente, isto é, não interfere no movimento geral-específico-geral que o texto como um todo deve realizar. Os Resumos de quase todas as áreas de estudo são escritos de uma maneira muito similar. Os tipos de informação incluídos e a ordem em que aparecem são muito convencionais, de modo que podem ser enunciados como modelos de Resumo, que visam guiar o escritor na escolha do tipo de informação que deve ser incluída no Resumo e da ordem que devem obedecer.

Levando em consideração essa estruturação de textos científicos, um corpus com 49 dissertações de mestrado e 3 teses de doutorado foi compilado. Sua análise superficial visou ao levantamento de padrões para que fosse feito um recorte das partes dos textos que seriam mais aprofundadas e, portanto, os alvos de maior atuação das ferramentas do sistema SciPo. Em um primeiro momento, decidiu-se investigar o Resumo, seguido das Introduções e Conclusões.

O protótipo do sistema SciPo está sendo implementado como uma aplicação Web, permitindo assim maior facilidade de acesso ao usuário, utilizando as linguagens PHP, Prolog, HTML (formatada por folhas de estilo CSS[1]) e JavaScript, além da ferramenta Jfor[2]. Na próxima seção serão detalhadas a arquitetura da aplicação, as técnicas utilizadas em sua implementação e a forma com que a estrutura de textos científicos estudada é utilizada para auxiliar o usuário no processo de escrita.

2. Arquitetura do Sistema

A implementação do sistema SciPo está sendo realizada num servidor FreeBSD 4.7-STABLE. A versão atual do projeto[3] pode ser acessada via um navegador Web, bastando para tanto que o navegador seja gráfico, possua um interpretador JavaScript (de no mínimo versão 1.3) instalado e habilitado e possa interpretar folhas de estilo CSS versão 2. A interação entre o usuário e a aplicação é proporcionada justamente pela utilização da linguagem JavaScript em conjunto com as páginas HTML geradas no servidor. Essas páginas baseiam-se na recomendação HTML versão 4.01 e no uso de folhas de estilo CSS. Do lado servidor, a aplicação faz uso de scripts PHP a fim de coordenar as requisições dos usuários, ativar programas Prolog e gerar as referidas páginas que serão exibidas no navegador do lado cliente. Atualmente, estão em uso no servidor o pré-processador de hipertexto PHP versão 4.2.3 e o compilador YAP Prolog versão 4.3.22. Adicionalmente, a ferramenta Jfor é utilizada para a transformação do texto redigido pelo usuário em um arquivo do formato RTF (*Rich Text Format*). Essa ferramenta, instalada em sua versão 0.7.1, necessita de uma *Java Virtual Machine* para funcionar.

A arquitetura atual do sistema SciPo está representada na Figura 1. Todos os recursos, processos e informações de entrada e saída nela representados, bem como a utilização de cada uma das ferramentas acima citadas, serão explicados nos próximos itens desta Seção.

---

[1] *Cascading Style Sheets*, folhas de estilo usadas para especificar como as informações numa página HTML são apresentadas.

[2] Software de código aberto desenvolvido por um grupo de voluntários para conversão de documentos XML (compatíveis com a especificação XSL-FO) para o formato RTF. *Website* do projeto: http://www.jfor.org , disponível em 25 de Julho de 2003.

[3] http://www.nilc.icmc.usp.br/~scipo

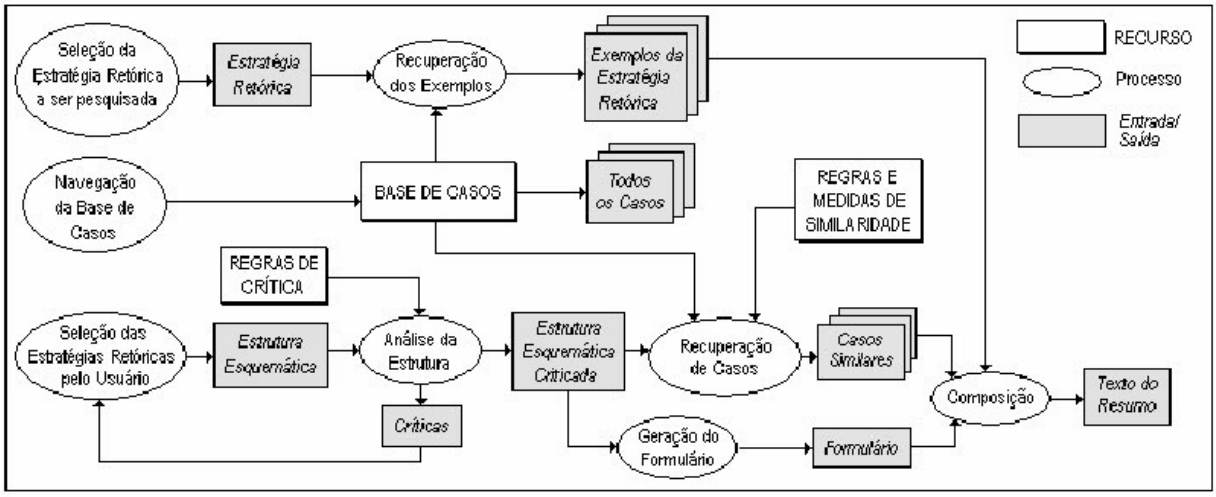

Figura 1 - Arquitetura atual do sistema SciPo

2.1. A Base de Casos

Como citado na seção introdutória desse relatório, um corpus com 52 textos foi compilado e analisado. Desse corpus, os Resumos, além das Introduções e Conclusões, foram etiquetados de forma a associar-lhes informações úteis. Para tal, foi proposto um conjunto de marcas, em XML, visando à marcação do corpus em quatro níveis: (1) componentes estruturais, (2) estratégias retóricas, (3) padrões de escrita e (4) marcadores discursivos. A marcação do corpus foi realizada manualmente em cada um dos 52 Resumos, considerando um esquema que descreve os componentes e estratégias retóricas previstas para um Resumo, apresentado na Figura 2.

1 Contexto	**4 Metodologia**
C1. Declarar proeminência do tópico	M1. Listar critérios ou condições
C2. Familiarização de termos	M2. Citar/Descrever materiais e métodos
C3. Introduzir a pesquisa a partir da grande área	M3. Justificar a escolha pelos materiais e métodos
2 Lacuna	**5 Resultado**
G1. Citar problemas/dificuldades	R1. Descrever o artefato
G2. Citar necessidades/requisitos	R2. Apresentar resultados
G3. Citar a ausência ou pouca pesquisa anterior	R3. Comentar/discutir resultados
3 Propósito	**6 Conclusão**
P1. Indicar o propósito principal	Co1. Apresentar conclusões
P2. Re-frasear/detalhar/especificar o propósito	Co2. Apresentar contribuições/valor do trabalho
P3. Introduzir mais propósitos	Co3. Apresentar recomendação

Figura 2 - Esquema utilizado para a marcação dos resumos

Para ilustrar a utilização desse esquema, é mostrado, na Figura 3, um exemplo de resumo codificado em XML. Nele é possível observar as anotações dos componentes estruturais, através da marca XML denominada **Subcomponente**, e também das estratégias retóricas, através da marca **Estrategia**. Nesse mesmo exemplo também estão anotados os padrões de escrita e os marcadores discursivos, respectivamente indicados através das marcas **Reutilizavel** e **Marcador**. Essa representação XML é utilizada, na implementação atual, para recuperar e exibir ao usuário exemplos reais de Resumos.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Resumo id="ES11" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="D:\
Resumos\schema_resumo.xsd">
        <Subcomponente>
                <Nome>Proposito</Nome>
                <Estrategia>
                        <Nome>Indicar o proposito principal (indicacao de
metodologia)</Nome><Reutilizavel>Um estudo empírico visando avaliar a </Reutilizavel>eficácia em revelar erros, a
dificuldade de satisfação e o custo de aplicação do critério Análise de Mutantes <Reutilizavel>é apresentado neste
trabalho</Reutilizavel>.</Estrategia>
                <Estrategia>
                        <Nome>Introduzir mais propositos</Nome> Eficácia e custo
<Reutilizavel><Marcador>também</Marcador> foram avaliados para os </Reutilizavel>critérios Potenciais-Usos,
<Reutilizavel>comparando-se <Marcador>assim</Marcador> os resultados obtidos para esses
</Reutilizavel>critérios.<Reutilizavel> A</Reutilizavel> especificação e implementação <Reutilizavel>de uma
</Reutilizavel>estratégia de minimização de conjuntos de casos de teste adequados ao critério Análise de Mutantes
<Reutilizavel><Marcador>também</Marcador> é apresentada</Reutilizavel>.</Estrategia>
        </Subcomponente>
        <Subcomponente>
                <Nome>Resultado</Nome>
                <Estrategia>
                        <Nome>Comentar/discutir (generalizar/explicar/comparar)
resultados</Nome><Reutilizavel>Através dos resultados obtidos observou-se que os </Reutilizavel>critérios Potenciais-
Usos (baseado em fluxo dados) e o critério Análise de Mutantes (baseado em erros) <Reutilizavel>são promissores
<Marcador>e</Marcador> apresentam </Reutilizavel>características complementares <Reutilizavel>que merecem ser
investigadas em</Reutilizavel> um experimento de maior porte.</Estrategia>
                <Estrategia>
                        <Nome>Apresentar resultados</Nome>  <Reutilizavel>A utilização de
</Reutilizavel>mutação restrita <Reutilizavel><Marcador>e</Marcador> de</Reutilizavel> minimização de conjunto de
casos de teste <Reutilizavel>constituem</Reutilizavel> mecanismos <Reutilizavel>que viabilizam a</Reutilizavel>
aplicação desses critérios em ambientes de produção de software. </Estrategia>
        </Subcomponente>
</Resumo>
```

Figura 3 - Exemplo de Resumo codificado em XML

No entanto, para realizar buscas e comparações apenas com componentes estruturais e estratégias retóricas, sem levar em consideração o texto propriamente dito do Resumo, é conveniente ter-se uma representação alternativa da base de casos. Essa tarefa de acesso à estrutura dos casos da base foi reaproveitada do ambiente AMADEUS, cujo Raciocínio Baseado em Casos foi desenvolvido em linguagem Prolog, possibilitando que toda a base de casos também fosse codificada nessa linguagem. A Figura 4 mostra o mesmo caso da Figura 3, agora representado como um predicado Prolog, identificado por **case**. Cada predicado desse tipo é guiado pela estrutura:

case(Nome_Caso, Lista_de_Componentes, Lista_de_Características_Pragmaticas).

No exemplo mostrado na Figura 4, o nome do caso é **r_es11** e a lista de componentes é composta por Propósito, duas vezes, e Resultado, também duas vezes. O último parâmetro da estrutura de um caso, **Lista_de_Características_Pragmaticas**, não está sendo utilizado neste projeto e é representado como uma variável anônima do Prolog (_). Cada um dos componentes na **Lista_de_Componentes** é representado pela estrutura **c(Nome_Componente, Estrategia)**, sendo que **Estrategia** tem a estrutura **s(Nome_Estrategia, Lista_de_Mensagens)**. Novamente um parâmetro, desta vez **Lista_de_Mensagens**, não é utilizado neste projeto e é substituído por uma variável anônima do Prolog.

```
case(r_es11,
[c(proposito,s(indicar_o_proposito_principal,_)),
c(proposito,s(introduzir_mais_propositos,_)),
c(resultado,s(comentar_discutir__generalizar_explicar_comparar__resultados,_)),
c(resultado,s(apresentar_resultados,_))],_).
```

Figura 4 - Exemplo de estrutura de Resumo codificada em Prolog

Na área do servidor em que o protótipo está sendo implementado, a base de casos de Resumos representada em XML está armazenada no subdiretório **resumos**. Já a base representada em Prolog está localizada no arquivo **Case_base_resumos.pro**.

2.2. Navegação da Base de Casos

Na interface do sistema existe uma opção no menu principal chamada **Navegação pela Base**, que permite ao usuário visualizar um a um os exemplos de Resumos da base de casos. Quando essa opção é acionada, o script **resumos_consulta_base.php.inc**, um script PHP, é requisitado para realizar uma consulta Prolog que retorna uma lista formada pelos nomes de todos os casos da base. Essa consulta é realizada executando-se o script Shell **resumos_consulta_base.sh**, o qual faz uma chamada ao predicado Prolog **consulta_base**[4], construído para montar a lista de casos existentes na base. As consultas Prolog foram construídas através de scripts Shell para que o compilador YAP Prolog não seja executado no modo interativo de consultas, pois o que se espera é que o compilador seja finalizado automaticamente após uma consulta ser realizada.

Ao receber a resposta proveniente da consulta Prolog, o script PHP mencionado percorre a lista de nomes de casos recuperados montando, para cada nome, um elemento de um objeto HTML do tipo **Select**. Assim, o que o usuário obtém ao escolher a opção **Navegação pela Base** é uma lista de nomes de casos dispostos dentro de um objeto gráfico rolável, do qual é possível visualizar qualquer um dos casos da base realizando um clique-duplo no item desejado ou acionando o botão **Ver Caso**.

Para que um caso particular seja exibido, a função JavaScript **exibe_caso**[5] é chamada, recebendo como parâmetro o nome do objeto HTML do tipo **Select**. O que essa função faz é exibir em uma nova janela o script **mostra_caso.php**, passando para ele o nome do caso a ser exibido por meio do método GET de envio de dados do protocolo HTTP, ou seja, o nome do caso é enviado junto à URL do script PHP. Nele, o caso escolhido é lido do respectivo arquivo XML através de um parser disponibilizado pela linguagem PHP, o qual permite que funções customizadas sejam definidas para serem ativadas assim que o parser identifica uma marca XML de início, uma marca de fim, ou um bloco de caracteres entre marcas. Essas funções recebem como parâmetro ou o nome da marca atual (de início ou de fim) e o valor de seus respectivos atributos, ou o bloco de dados lido entre as marcas, dependendo do contexto em que a função é utilizada, possibilitando que o texto do Resumo seja formatado pelo script

[4] Todos os predicados Prolog citados neste relatório, senão o predicado **case**, estão localizados no arquivo **scipo.pro**.

[5] Todas as funções JavaScript, exceto as referentes à fase de críticas, estão contidas no arquivo **funcoes.js**.

de acordo com as marcas XML que vão sendo identificadas pelo parser. Na Figura 5 é possível observar o mesmo caso da Figura 3 formatado pelo script **mostra_caso.php**, o qual exibe os padrões de escrita destacados em negrito e os marcadores discursivos numa cor diferente do restante do texto.

2.3. Navegação dos Marcadores Discursivos

É possível acessar uma lista completa de marcadores discursivos agrupados por tipo de sinalização, representativos das bases de casos de Resumos, Introduções e Conclusões. Essa lista está implementada no arquivo **consulta_marcadores.htm.inc**, um arquivo HTML estático que apresenta logo em seu início um índice de tipos de sinalização, permitindo escolher qual grupo de marcadores discursivos será exibido.

2.4. Exemplos de Estratégia Retórica

Também é disponibilizada ao usuário a opção de visualizar todos os exemplos de determinada estratégia retórica contidos na base de casos, por meio da opção **Exemplos de Estratégias**, localizada no menu principal do aplicativo. Para tal, o sistema exibe uma lista de estratégias, baseada na Figura 2, permitindo ao usuário selecionar apenas um elemento por vez. É realizada então uma consulta Prolog que retorna uma lista de nomes de casos que fazem uso da estratégia escolhida, para, enfim, exibir os trechos desses casos referentes à seleção do usuário.

De certo modo, a implementação desse recurso é semelhante à implementação da **Navegação pela Base**, explicada na Seção 2.2. Em ambas as opções, um predicado Prolog é executado e o seu resultado indica os nomes dos casos que serão lidos pelo parser XML do PHP. No entanto, existem duas diferenças relevantes na implementação da exibição de exemplos de estratégia retórica, implementada no script **resumos_consulta_estr.php.inc**. Em primeiro lugar, é preciso fornecer ao programa Prolog qual estratégia deve ser considerada no momento da busca. Para isso, o script PHP mencionado escreve num arquivo, chamado **features.txt**, a estratégia retórica escolhida juntamente com seu respectivo componente estrutural. Assim, ao executar o predicado **consulta_estr**, por meio do script Shell

resumos_consulta_estr.sh, o arquivo **features.txt** é lido e a lista de casos que fazem uso da estratégia selecionada pelo usuário é computada.

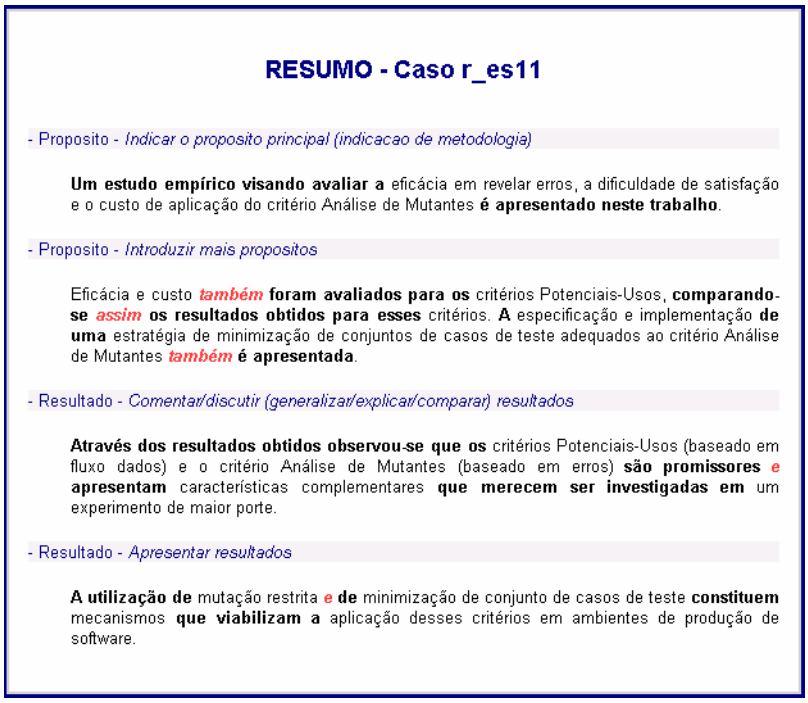

Figura 5 - Resumo em XML formatado por um script PHP

A segunda diferença refere-se à utilização do parser XML. Dessa vez não é necessário exibir o Resumo por completo, e sim apenas os trechos relacionados à estratégia retórica selecionada, o que implica numa alteração das funções customizáveis, citadas na Seção 2.2, usadas em conjunto com o parser XML. Essas funções foram reconstruídas de modo que seja possível identificar qual a estratégia retórica lida mais recentemente pelo parser e também selecionar qual porção de texto deve ser exibida ao usuário. Na interface foi adicionada uma opção para exibir o caso completo referente a determinada porção de texto em exibição, utilizando novamente o script **mostra_caso.php** explicado na Seção 2.2, dessa vez por meio da função JavaScript **exibe_caso2**.

No mesmo script PHP que exibe os exemplos de estratégias retóricas, é implementado o recurso de inserção de trechos de textos reutilizáveis, utilizado na fase de composição do

Resumo. Como esse recurso não está disponível nas outras partes do sistema, inclusive na parte discutida nesta Seção, ele somente será explicado na Seção 2.4.3.

2.5. Redação do Resumo

Após o usuário familiarizar-se com o esquema proposto de marcação de Resumos, através do contato direto com os exemplos de textos reais e suas respectivas classificações, ele se torna mais apto a construir seu próprio Resumo usando os componentes estruturais e as estratégias retóricas mais apropriadas às suas necessidades. O que o sistema faz é fornecer uma lista, novamente baseada na Figura 2, em que é possível selecionar a quantidade e a ordem das estratégias desejadas, para, logo a seguir, avaliar se a escolha do usuário é coerente. Dada uma lista de estratégias já avaliada, o aplicativo pode fornecer casos com estrutura semelhante à escolhida pelo usuário ou ainda pode disponibilizar áreas para composição de Resumo, nas quais é possível inserir trechos padrões de texto encontrados na base de casos. Cada um desses recursos será explicado em maiores detalhes no decorrer desta Seção.

2.5.1. Seleção e Análise da Estrutura

A interface disponibilizada ao usuário para escolha das estratégias retóricas, implementada no arquivo **resumos_redacao_estr.php**, está reproduzida na Figura 6. Nessa interface é permitido selecionar qualquer estratégia, em qualquer quantidade e em qualquer ordem. É possível também alterar a ordem das estratégias e excluir uma ou todas as estratégias selecionadas, as quais estão contidas na área da interface denominada **Estratégias Escolhidas**. Todas essas funcionalidades foram implementadas utilizando a linguagem JavaScript.

Durante a fase de seleção de estratégias, é possível acessar o sistema de críticas da aplicação. As regras de crítica foram formuladas de forma a tentar corrigir padrões de estruturação considerados problemáticos em comparação com os modelos prescritos pela literatura. As críticas consideram desvios de conteúdo (falta de estratégias consideradas essenciais) e desvios de ordem (ordem de ocorrência das estratégias dentro da estrutura).

Dessa forma, existem quatro classes de regras: críticas de conteúdo, críticas de ordem, sugestões de conteúdo e sugestões de ordem.

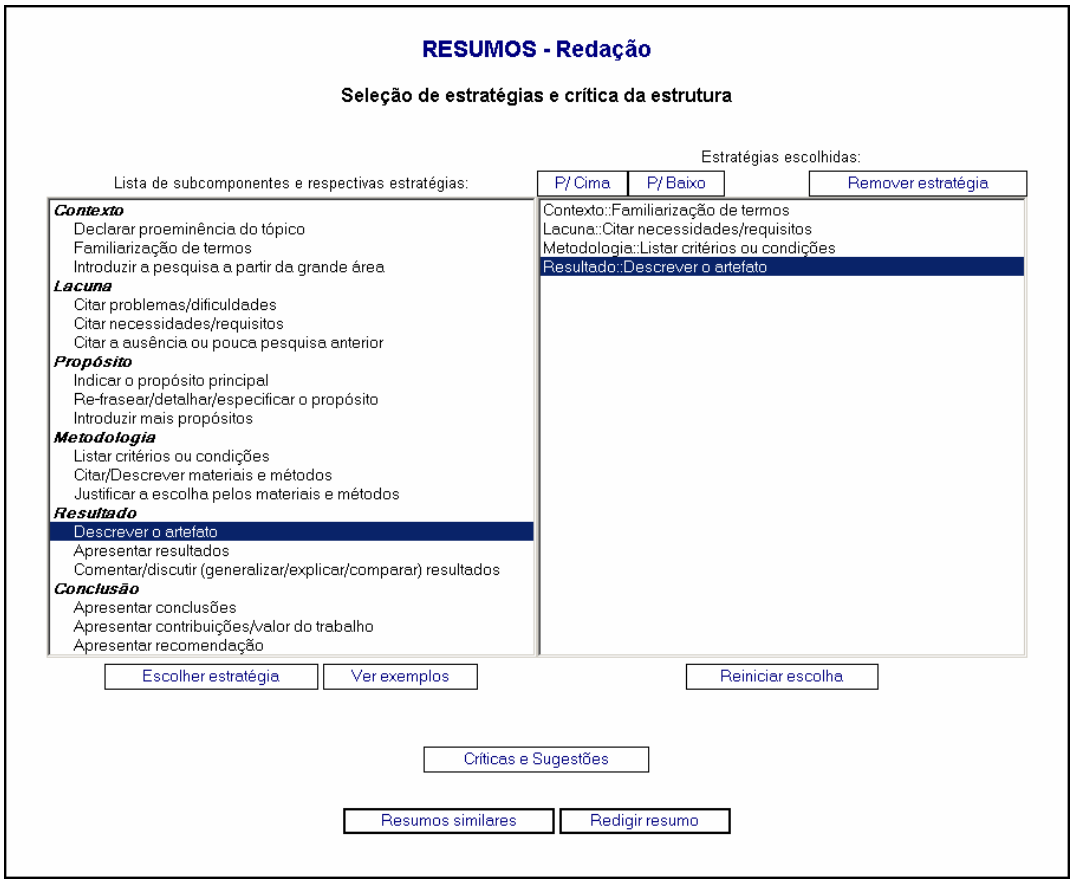

Figura 6 - Interface usada na seleção de estratégias

Quando o sistema de críticas é acionado, por meio do arquivo **resumos_criticas.php**, a função JavaScript **critica_selecao** é executada para analisar a estrutura montada pelo usuário e exibir mensagens relacionadas aos desvios nela encontrados. Essa função aplica na estrutura escolhida todas as regras de cada uma das quatro classes citadas, a fim de verificar se alguma é violada, para assim exibir mensagens que orientam o usuário no sentido de aprimorar sua escolha. Todas as regras e mensagens estão descritas no Anexo 1. Quando alguma regra das classes de crítica de conteúdo ou de crítica de ordem for quebrada, não é permitido ao usuário seguir adiante no processo de redação de seu Resumo, ou seja, só é possível acessar as outras funcionalidades da fase de redação quando, ou nenhuma regra for

violada, ou somente regras das classes de sugestões foram quebradas. A Figura 7 mostra o resultado da análise da estrutura reproduzida na Figura 6.

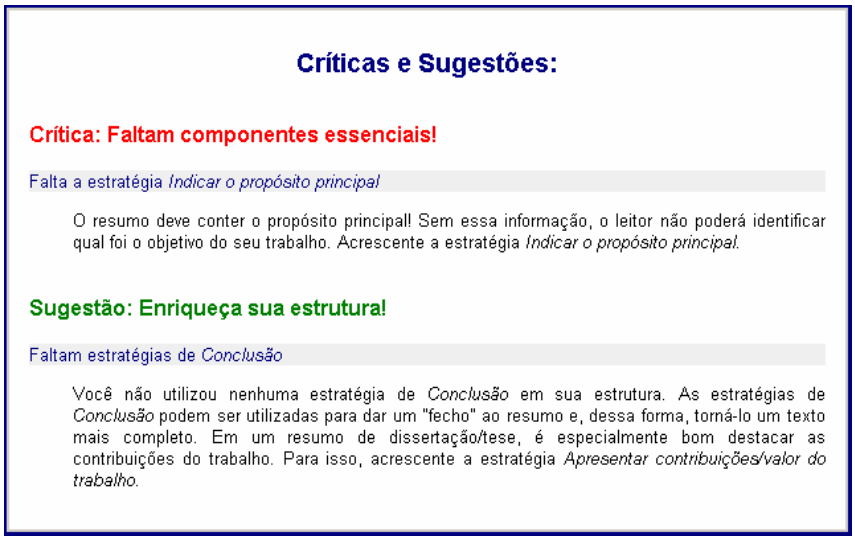

Figura 7 - Exemplo de críticas e sugestões

2.5.2. Recuperação de Casos Similares

Os casos com estrutura similar à requisição feita pelo usuário são recuperados através de um método baseado em regras de similaridade entre listas (casamento de padrões) e na medida de similaridade conceitual chamada *nearest neighbors matching*. A implementação da busca por casos similares no projeto SciPo foi baseada na implementação dessa mesma funcionalidade no projeto AMADEUS. Naquele projeto, a estrutura dos casos, codificados como predicados Prolog, é comparada com a estrutura escolhida pelo usuário (chamada de requisição e representada como uma lista também da linguagem Prolog) utilizando conceitos de comparações entre listas. Foram usados três tipos dessas comparações: igualdade, inclusão e intersecção não-nula. Desses três tipos, foram derivados quatro predicados que tratam das seguintes situações:

- *Requisição totalmente contida no caso*: Os casos iguais à requisição aqui são desconsiderados. São recuperados os casos que contêm totalmente a requisição, cujos elementos não podem estar dispostos de maneira diferente no caso. Por exemplo, se tivermos a lista hipotética [a,b,c,d,e], ela não estará contida na lista [a,1,b,c,2,d,e], mas

estará na lista [1,2,a,b,c,d,e]. Implementada através do predicado **similar1**, tem o valor de similaridade, um número entre 0 e 1 posteriormente multiplicado por 100, dado pela divisão do tamanho da requisição pelo tamanho do caso.

- *Requisição parcialmente contida no caso*: Aqui são consideradas intersecções não-nulas entre caso e requisição, desde que o tamanho da requisição seja menor ou igual ao tamanho do caso. Casos que são idênticos à requisição, ou a contém totalmente são desconsiderados. Vale ressaltar que a intersecção retornada, por exemplo, entre as listas [h,i,j,l,m,n,o] e [i,h,j,l,m,o,n], é a lista [j,l,m]. É implementada no predicado **similar2** e o valor de similaridade é dado pela divisão do tamanho da intersecção pela soma entre o tamanho da requisição, o tamanho do caso e o valor negativo do tamanho da intersecção.

- *Caso totalmente contido na requisição*: implementada no predicado **similar3**, é análoga a **similar1**. Aqui, o valor de similaridade é calculado pela divisão do tamanho do caso pelo tamanho da requisição.

- *Caso parcialmente contido na requisição*: é análoga à segunda situação, com a ressalva de que o tamanho do caso deve ser menor que o tamanho da requisição. Implementada através do predicado **similar4**, tem o valor de similaridade computado da mesma maneira que **similar2**.

Com essa base construída no projeto AMADEUS em mãos, foi proposta uma ampliação dos recursos utilizados na busca de casos similares, através de uma variação do que já havia sido feito anteriormente. Essa proposta, baseada em operações da Teoria dos Conjuntos, levou a uma nova implementação dos algoritmos de comparação entre estruturas retóricas. Foram criados novos predicados [3] para tratar dos testes de igualdade, inclusão e intersecção não-nula entre conjuntos, dos quais se derivou mais quatro testes de comparação análogos aos anteriormente citados. Esses novos testes verificam as situações de requisição totalmente contida no caso, requisição parcialmente contida no caso, caso totalmente contido na requisição e caso parcialmente contido na requisição, desconsiderando a ordem e a quantidade dos elementos nas listas, conforme estabelecido na Teoria dos Conjuntos. Cada uma dessas situações é implementada, respectivamente, nos predicados **similar1_semord**, **similar2_semord**, **similar3_semord** e **similar4_semord**, descartando-se os casos que seriam recuperados pelos respectivos predicados análogos de comparação entre listas, a fim de se

evitar respostas repetidas. O cálculo do valor de similaridade nesses novos predicados é feito da mesma maneira que nos predicados anteriores, desconsiderando dessa vez elementos repetidos das listas.

A partir dessas oito estratégias de busca, mais os testes de igualdade entre listas e entre conjuntos, foram montados quatro níveis de recuperação de casos similares:

- *1º Nível*: Casos que contêm toda a requisição, levando em consideração a ordem das estratégias. Utiliza o teste de igualdade entre listas, mais o predicado **similar1**.

- *2º Nível*: Casos que contêm parte da requisição, levando em consideração a ordem das estratégias. Utiliza o resultado dos testes **similar2**, **similar3** e **similar4**.

- *3º Nível*: Casos que contêm toda a requisição, desconsiderando a ordem das estratégias. Utiliza o teste de igualdade entre conjuntos, e o predicado **similar1_semord**.

- *4º Nível*: Casos que contêm parte da requisição, desconsiderando a ordem das estratégias. Faz uso dos testes **similar2_semord**, **similar3_semord** e **similar4_semord**.

A apresentação ao usuário dos casos recuperados, usando esses níveis, está a cargo do script **resumos_redacao_casos.php.inc**. As consultas Prolog são feitas de maneira semelhante à realizada na fase de visualização de exemplos de estratégia retórica, explicada na Seção 2.3. Um arquivo texto (**features.txt**) é montado com a lista de estratégias retóricas e respectivos componentes estruturais da requisição. Desse arquivo, os predicados **todas_estrat**, **algumas_estrat**, **todas_estrat_semord** e **algumas_estrat_semord**, os quais implementam respectivamente cada nível de recuperação de casos, montam a lista Prolog que representará a requisição do usuário nos testes de comparação subseqüentes. Em cada nível, é possível descartar casos abaixo de um determinado valor de similaridade, através do predicado **retira_menores**. No 2º nível, casos com valor de similaridade abaixo de 25% são descartados. No 4º nível, o valor limite é 40%. Todos os casos recuperados nos 1º e 3º níveis são exibidos. No Anexo 2, a implementação de todos os predicados Prolog utilizados na recuperação de casos similares é apresentada.

Os níveis de recuperação de casos são exibidos ao usuário gradativamente: de início, somente o primeiro nível é exibido, deixando os outros níveis para serem posteriormente exibidos um a um, através da opção **Recuperar Mais Casos**. Para cada nível, são exibidos o nome do caso recuperado e seu respectivo valor de similaridade em relação à requisição, como mostra a Figura 8. O programa PHP **resumos_redacao_casos.php.inc** executa cada nível através de quatro scripts Shell diferentes, e ainda permite que os casos recuperados sejam visualizados por completo através de um clique-duplo sobre o nome do caso, utilizando a implementação já discutida na Seção 2.2. Cabe ainda ressaltar que, nessa fase, é permitido ao usuário retornar à fase de seleção de estratégias retóricas para alterar a requisição previamente escolhida.

2.5.3. Formulário de Composição

A criação do Resumo pode ser realizada em um formulário de composição gerado pelo sistema através do script **resumos_redacao.php**. Nesse formulário, é criada uma caixa de texto para cada estratégia retórica selecionada pelo usuário, de modo que a construção do Resumo seja orientada à estrutura escolhida. Para cada caixa de texto existem dois contadores, um para o número de caracteres e outro para o número de palavras digitadas, implementados em JavaScript, e que serão usados posteriormente num controle de balanceamento do texto.

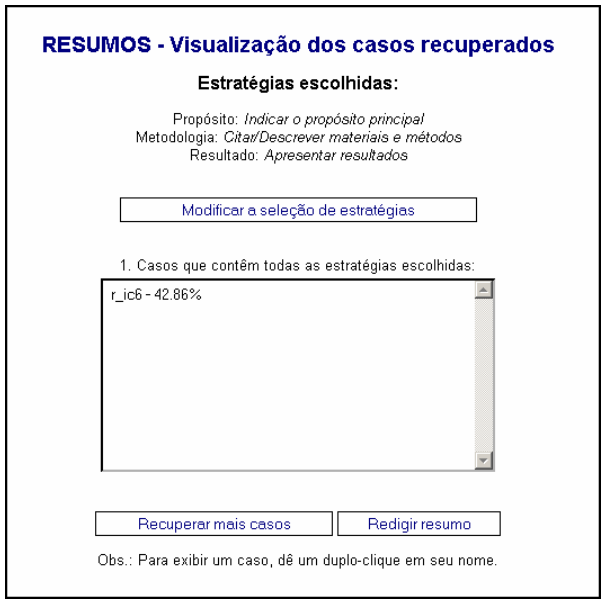

Figura 8 - Casos recuperados pelo primeiro nível de busca

Como as caixas de texto estão associadas a estratégias retóricas, é fornecida ao usuário a possibilidade de visualizar diretamente exemplos de cada uma dessas estratégias através dos *links* denominados **Ver Exemplos**. Nesse caso, é utilizado o script **resumos_consulta_estr.php.inc**, explicado inicialmente na Seção 2.3. Para esse novo contexto, é adicionado o recurso de inserção, nas caixas de texto, de trechos de texto reutilizáveis (os quais estão anotados na base de Resumos XML pela marca denominada **Reutilizável**). O script PHP, além de exibir os Resumos, cria trechos de texto que contêm apenas as palavras que estão anotadas pela marca **Reutilizável**, substituindo os trechos não reutilizáveis por lacunas. A implementação ainda separa os trechos por sentença e identifica de qual caixa de texto o usuário selecionou a opção **Ver Exemplos**. Utilizando JavaScript, tornou-se possível inserir um determinado trecho reutilizável na caixa de texto apropriada, como ilustra a Figura 9.

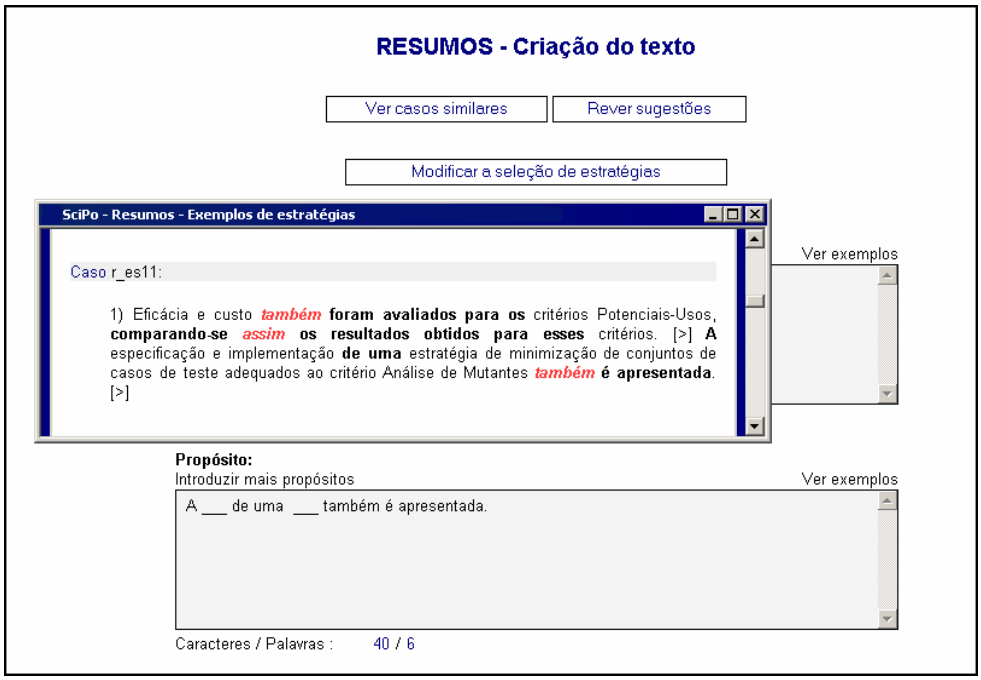

Figura 9 - Exemplo de inserção de texto reutilizável

Do Formulário de Composição é possível visualizar casos similares (conforme Seção 2.4.2), ver sugestões de conteúdo ou de ordem quanto à estrutura construída (conforme Seção 2.4.1) e ainda retornar à interface de seleção de estratégias para realizar alterações na estrutura

já avaliada pelo sistema. Na implementação dessas alterações foi necessário manter o texto que o usuário possa já ter criado, para que, ao voltar ao formulário de composição, não se perca o trabalho de redação já realizado. A solução utilizada foi enviar o texto digitado ao servidor através de um formulário HTML. O servidor, por sua vez, cria a próxima página de forma que ela contenha o texto redigido de modo transparente ao usuário. Esse processo continua até que o formulário de composição seja requisitado novamente, quando o texto armazenado volta a ser disposto nas caixas de texto apropriadas, tornando-se novamente visível ao usuário. Uma outra alternativa seria armazenar o texto no servidor para futuramente ser recuperado. Nesse caso, existe a necessidade de se manter várias sessões para vários usuários que possam estar utilizando o sistema ao mesmo tempo. Como a linguagem PHP tem um bom suporte a controle de sessões, a escolha entre uma ou outra alternativa deveu-se ao fato de que um controle de usuários não é necessário nessa implementação.

Por fim, na opção **Agrupar em Arquivo**, é possível transformar o texto redigido em um arquivo do formato RTF, que pode ser copiado para o computador do usuário para armazenamento e futuras modificações. Essa funcionalidade está implementada no arquivo PHP **gera_rtf.php**. Ele recebe o texto digitado contido em um só bloco de texto, agrupado pela função JavaScript **junta_para_rtf**, e gera um arquivo XSL-FO, chamado **resumo.fo**, que contém as especificações do documento a ser gerado mais o próprio bloco de texto. Um exemplo desse arquivo é mostrado na Figura 10. O próximo passo é executar a ferramenta Jfor para ler o arquivo XSL-FO e gerar um arquivo no formato RTF, pronto para ser copiado pelo usuário.

2.6. Considerações Adicionais

Procurou-se manter o sistema compatível com as versões mais recentes dos navegadores Web mais populares. O sistema foi testado nos navegadores Netscape Navigator 7 e Internet Explorer 6, sem apresentar problemas com relação a interpretações incorretas das páginas geradas pela ferramenta.

Praticamente toda a aparência da interface está definida em CSS, no arquivo **geral.css**. Lá estão especificados atributos como cores, fontes, bordas e alinhamentos. O cabeçalho, o rodapé e o menu das páginas estão em arquivos separados, e são lidos pelos scripts PHP no

momento da montagem das páginas. O cabeçalho está especificado no arquivo **header.html.inc**, o rodapé em **footer.html.inc** e os menus nos arquivos do tipo **menu_*.html.inc**. No Anexo 3 se encontra uma lista de todos os arquivos utilizados na implementação, acompanhados de uma breve descrição.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
	<fo:layout-master-set>
		<fo:simple-page-master master-name="my-page">
			<fo:region-body margin="1in"/>
		</fo:simple-page-master>
	</fo:layout-master-set>
	<fo:page-sequence master-reference="my-page">
		<fo:flow flow-name="xsl-region-body" font-size="12pt" font-family="Times">
			<fo:block font-weight="bold" font-size="16pt" font-family="Helvetica" space-after="12pt">
				Resumo
			</fo:block>
			<fo:block line-height="1.5" text-align="justify">
				Aqui vai o texto do Resumo.
			</fo:block>
		</fo:flow>
	</fo:page-sequence>
</fo:root>
```

Figura 10 - Exemplo de arquivo XSL-FO usado para criação de arquivos RTF

O controle de acessos simultâneos é feito, em parte, mantendo-se todas as informações de estado de utilização do sistema no lado cliente da arquitetura. Para exemplificar, existe a lista de estratégias retóricas selecionadas e o texto digitado pelo usuário, os quais ficam armazenados nas páginas geradas pela aplicação e são enviados novamente ao servidor a cada página da fase de redação que é requisitada. A Figura 11 ilustra essa abordagem. Além disso, determinados arquivos que são alterados durante a utilização do sistema e necessitam ser armazenados no servidor são protegidos pela função **flock** do PHP. Esses arquivos são o **features.txt**, usado nas consultas Prolog, e o **resumo.fo**, utilizado na geração de arquivos RTF. O bloqueio desses arquivos é mostrado na Figura 12. A única exceção é o arquivo RTF criado na fase de redação, o qual pode ser acessado num momento posterior ao da sua criação, de acordo com a vontade do usuário. Esse arquivo é criado no servidor e é disponibilizado ao

usuário para *download*, o que não implica que será imediatamente copiado para o computador cliente. Portanto, se esse tipo de arquivo for bloqueado, nenhum outro usuário poderá criar outro arquivo até que o primeiro usuário resolva usá-lo e liberá-lo. Essa particularidade é tratada do seguinte modo: sempre que um novo Resumo for gravado em RTF, é verificado se, dentre os arquivos RTF já gravados, existe algum que fora modificado há mais de 30 minutos. Se sim, esse arquivo será sobrescrito; senão, outro arquivo será criado para ser disponibilizado ao usuário. A Figura 12 ilustra o caso em que não haviam no servidor resumos gravados em RTF, e duas solicitações para gravação são atendidas consecutivamente, implicando na criação de dois arquivos diferentes.

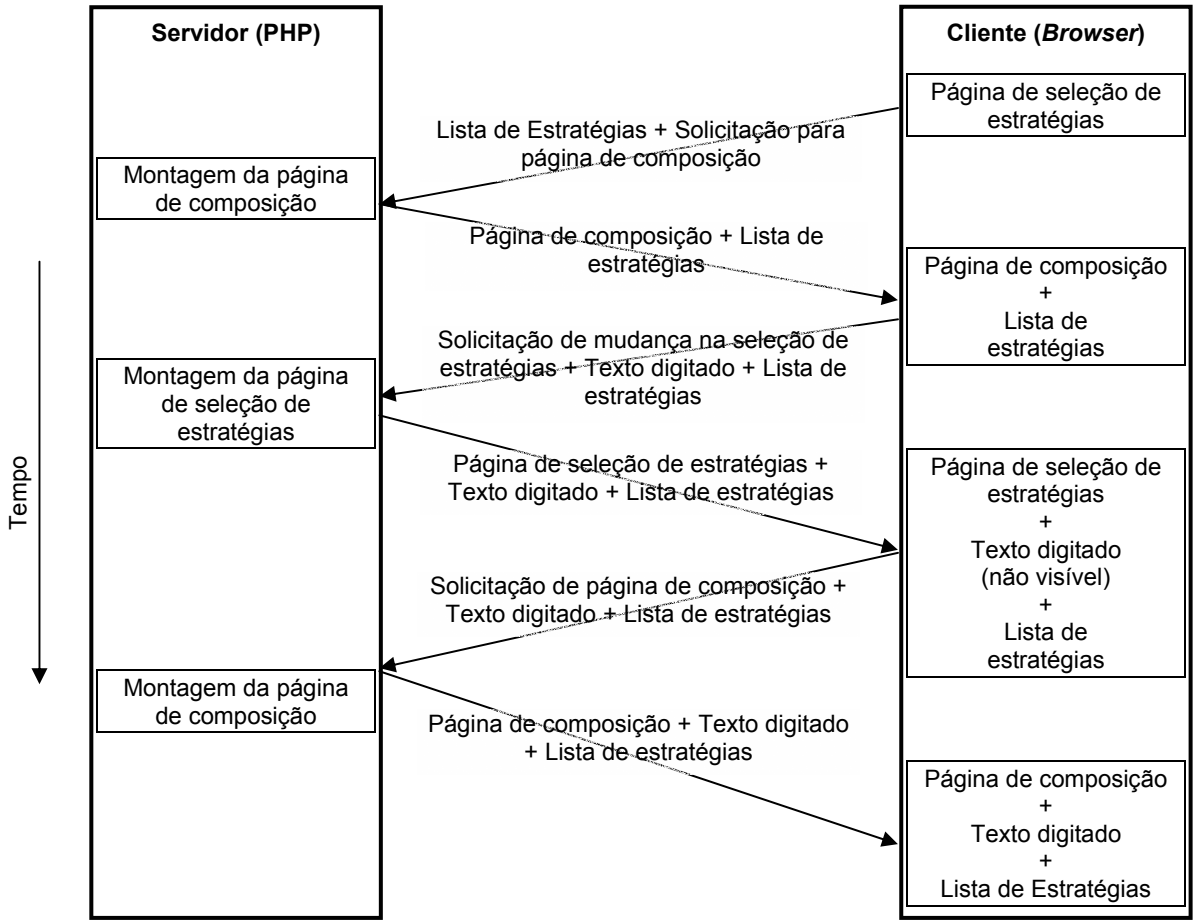

Figura 11 - Exemplo de informações de estado de utilização do sistema mantidos no lado cliente

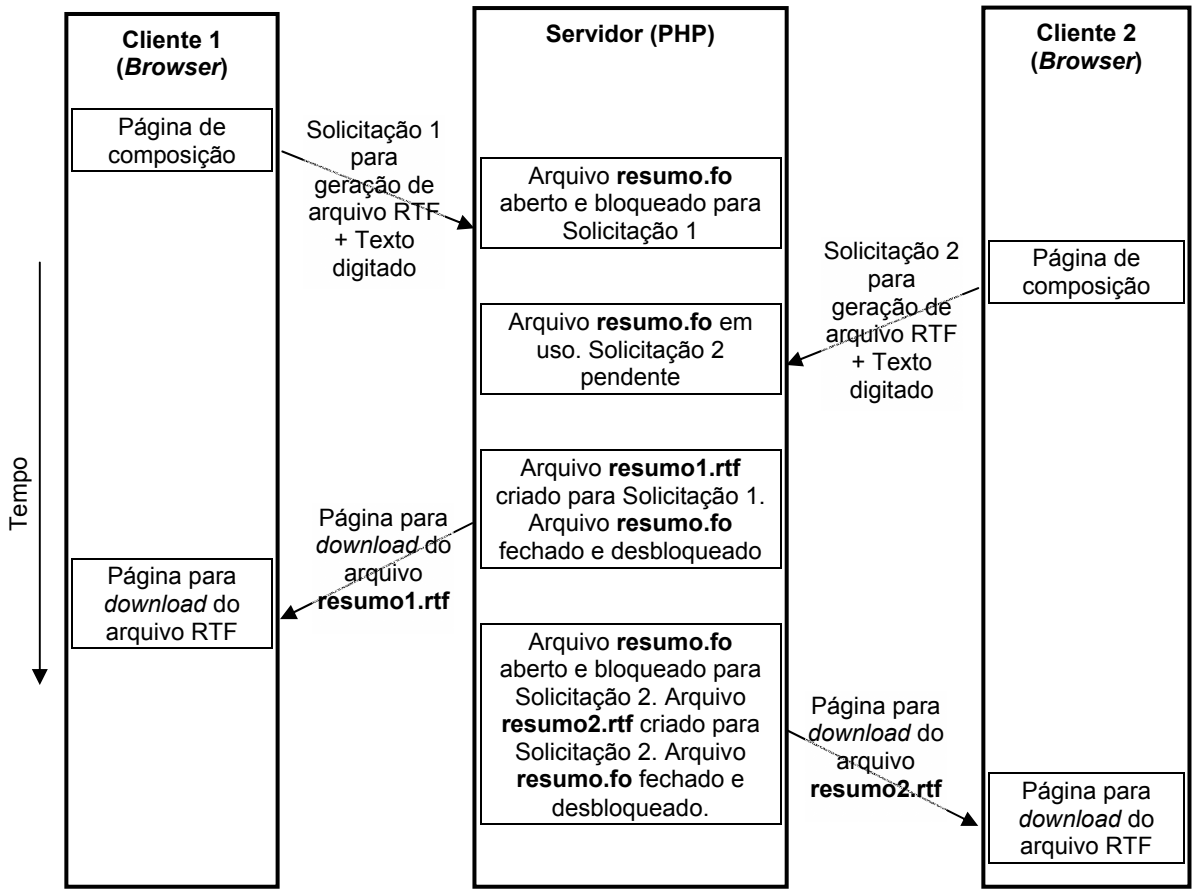

Figura 12 - Esquema de bloqueio de arquivos no servidor e geração de arquivos RTF

2.7. Futuras Expansões do Sistema

A base de Resumos aceita novos casos sem que seja necessário alterar o que já foi implementado. Os Resumos etiquetados em XML podem ser copiados diretamente para o diretório **resumos**. Além disso, basta criar um predicado Prolog para cada Resumo no arquivo **Case_base_resumos.pro**, nos moldes do exemplo da Figura 4. A partir daí, todas as consultas Prolog levarão em consideração a adição realizada na base de casos.

Para que o sistema passe a trabalhar com Introduções e Conclusões pode-se realizar modificações no código de modo que não seja necessário replicar toda a implementação. Os trechos de código que lidam especificamente com os Resumos podem ser copiados e alterados para manipular Introduções e Conclusões. Assim, algum parâmetro setado indiretamente pelo usuário no início da utilização da ferramenta indicará se os trechos de códigos específicos serão executados para Resumos, Introduções ou Conclusões.

O primeiro menu apresentado ao usuário, implementado no arquivo **menu_inicial.html.inc**, apresenta as seguintes opções: Resumos, Introduções e Conclusões. Atualmente somente a primeira opção leva a outra página, chamada **resumos_entrada.php**. Sugere-se que, para cada uma das opções, a página **resumos_entrada.php** seja chamada da seguinte maneira:

resumos_entrada.php?componente=resumo
resumos_entrada.php?componente=introducao
resumos_entrada.php?componente=conclusao

Desse modo, o script PHP **resumos_entrada.php** receberia a variável **componente** por meio do método GET de envio de dados do protocolo HTTP, e a leria assim

```
<?php $componente = $_REQUEST["componente"]; ?>
```

disponibilizando a variável **componente** por todo o script PHP. Analogamente, todas as outras páginas do sistema deveriam receber essa variável. Percebe-se também que o arquivo **resumos_entrada** poderia ter sido nomeado de outra forma, sem o prefixo "resumos", pois, como foi visto, ele pode ser utilizado para lidar também com Introduções e Conclusões. Esse tipo de identificação de arquivos foi criado no início do projeto, mas pode ser alterado, tomando as devidas precauções para alterar também os arquivos que fazem referência às páginas que forem renomeadas.

Como exemplo de utilização dessa variável, observa-se a montagem da lista de estratégias retóricas, realizada pelo script **resumos_redacao_estr.php**, para seleção da estrutura no início da fase de redação

```
<?php include('resumos_lista_estr.html.inc'); ?>
```

A lista de estratégias retóricas é carregada do arquivo **resumos_lista_estr.html.inc**. Se existirem dois outros arquivos análogos a esse, chamados **introducoes_lista_estr.html.inc** e **conclusoes_lista_estr.html.inc**, o seguinte trecho de código PHP carregaria a lista correta de acordo com o conteúdo da variável **componente**

```
<?php
    if ($componente == "resumo")
            include('resumos_lista_estr.html.inc');
    else if ($componente == "introducao")
            include('introducoes_lista_estr.html.inc');
    else if ($componente == "conclusao")
```

```
        include('conclusoes_lista_estr.html.inc');
?>
```

Esse tipo de teste também pode ser usado nas consultas Prolog, onde o que deve ser alterado é a base de casos para cada consulta, deixando os algoritmos de busca intactos. Para ilustrar, temos a consulta Prolog usada na opção de visualização de exemplos de estratégias, realizada pelo script Shell **resumos_consulta_estr.sh**, reproduzido abaixo

```
#!/usr/local/bin/yap -L
:- consult('scipo.pro'), consulta_estr(L), write(L).
```

Este script carrega o arquivo **scipo.pro**, por meio do predicado YAP **consult**, para enfim realizar a busca por casos, utilizando o predicado **consulta_estr**. Esse predicado por sua vez carrega a base de resumos do arquivo **Case_base_resumos.pro**, também utilizando o predicado **consult**, como pode ser verificado no Anexo 2. Para deixar o arquivo **scipo.pro** totalmente independente da base de casos em uso, essa parte de acesso à base pode ser movida para o arquivo **resumos_consulta_estr.sh**, o qual ficaria assim

```
#!/usr/local/bin/yap -L
:- consult('Case_base_resumos.pro'), consult('scipo.pro'),
consulta_estr(L), write(L).
```

Conseqüentemente, se forem criados mais dois scripts análogos ao **resumos_ consulta_estr.sh**, chamados **introducoes_consulta_estr.sh** e **conclusoes_consulta_estr.sh**, o arquivo **resumos_consulta_estr.php.inc**, que executa a consulta Prolog para exibição de exemplos de estratégias, poderia chamar os scripts Shell de acordo com o valor da variável **componente** discutida acima. Cabe aqui ressaltar que cada base de casos deve ter seu respectivo arquivo Prolog. Assim existiriam também os arquivos **Case_base_ introducoes.pro** e **Case_base_conclusoes.pro**.

Uma última observação refere-se a base de casos em XML, cujo acesso também deve ser modificado. Os casos são exibidos utilizando-se duas funções JavaScript, localizadas no arquivo **funcoes.js** e identificadas por **exibe_caso** e **exibe_caso2**. Essas funções enviam o nome do caso a ser exibido ao script PHP **mostra_caso.php**, novamente pelo método GET do protocolo HTTP. Esse script acessa o arquivo XML do diretório **resumos** e o exibe. Como todas as outras páginas do sistema, **mostra_caso.php** também deve receber a variável **componente** para saber de que diretório lerá o arquivo XML (aqui considera-se que existiriam mais dois diretórios, chamados **introducoes** e **conclusoes**, para armazenar as outras

duas bases de casos). Cabe às duas funções JavaScript mencionadas acima enviar ao script **mostra_caso.php** a variável **componente**. Elas deveriam ser alteradas para receber um novo parâmetro, e todos os arquivos que usam essas funções (**resumos_consulta_base.php.inc**, **resumos_consulta_estr.php.inc** e **resumos_redacao_casos.php.inc**) deveriam fornecer a elas o conteudo dessa variável.

3. Conclusões

Devido ao caráter distribuído da aplicação, foi necessário trabalhar com várias ferramentas de desenvolvimento, algumas propícias ao processamento no lado cliente, outras voltadas à execução no lado servidor. Dada essa variedade de linguagens e formatos, procurou-se deixar bem claro o papel de cada ferramenta já na implementação, separando códigos de linguagens diferentes em lugares diferentes. Existe uma exceção à essa segregação: os códigos HTML e PHP estão de certa forma unidos, devido à própria natureza pré-processadora de hipertexto da linguagem PHP.

A utilização do Raciocínio Baseado em Casos, herdado do projeto AMADEUS, demandou um maior esforço no entendimento das técnicas anteriormente codificadas em linguagem Prolog. Este estudo proporcionou, além da reutilização de grande parte do que fora implementado, a experimentação de variações do que já havia sido proposto naquele projeto.

Outras funcionalidades serão adicionadas ao sistema. Será realizado um tratamento do texto redigido no formulário, através da detecção do tempo verbal utilizado em cada estratégia e das correções ortográfica e gramatical. Haverá também um controle de balanceamento do texto, a fim de equilibrar as quantidades de texto redigidas em cada entrada do formulário. A interface Web do aplicativo também deverá ser aprimorada, fazendo parte de ações que, gradativamente, estão compondo uma ferramenta de auxílio à escrita científica em língua portuguesa.

4. Referências Bibliográficas

[1] Feltrim, V.D. *Suporte Computacional à Escrita Científica em Português*. Monografia de Qualificação, ICMC-USP, Março de 2002.

[2] Aluísio, S.M. *Ferramentas de Auxílio à Escrita de Artigos Científicos em Inglês como Língua Estrangeira*. PhD Thesis, IFSC-USP, Agosto de 1995.

[3] Monard, M.C.; Nicoletti, M.C. *Programas Prolog para Processamento de Listas e Aplicações*. Notas Didáticas, ICMC-USP, Janeiro de 1993.

Anexo 1 – Lista de Regras de Crítica e Mensagens

1ª Classe de regras – Críticas de conteúdo

Mensagem padrão: ***"Faltam componentes essenciais!"***

Mensagens específicas:

1. Falta a estratégia *Indicar o propósito principal*
    - "O resumo deve conter o propósito principal! Sem essa informação, o leitor não poderá identificar qual foi o objetivo do seu trabalho. Acrescente a estratégia *Indicar o propósito principal*."
2. Faltam estratégias de *Metodologia* (nenhuma das 3 foi utilizada)
    - O resumo deve indicar objetivamente ao leitor a metodologia empregada para a realização do seu trabalho. Acrescente pelo menos uma das 3 estratégias de metodologia. Escolha a que for mais adequada ao seu resumo.
3. Faltam as estratégias *Descrever o artefato* e *Apresentar resultados* (nenhuma das 2 foi utilizada)
    - O resumo deve apresentar ao leitor os principais resultados do seu trabalho. Acrescente a estratégia *"Descrever o artefato"* caso você queira enfocar o desenvolvimento de um artefato computacional como o principal resultado de seu trabalho. Acrescente a estratégia *"Apresentar resultados"* caso seus principais resultados sejam outros.

2ª Classe de regras – Críticas de ordem

Mensagem padrão: ***"Reorganize a estrutura!"***

Mensagens específicas:

1. A estratégia *Indicar o propósito principal* aparece **mais de uma vez**
    - O propósito principal do seu trabalho deve ser enunciado uma única vez e de forma clara e objetiva. Portanto, deixe apenas uma ocorrência da estratégia *Indicar o propósito principal* em sua estrutura. Caso queira fornecer mais informações sobre o propósito, acrescente a estratégia *Re-frasear/detalhar/especificar o propósito*. Caso você queira acrescentar outros propósitos além do principal (propósitos secundários), utilize a estratégia *Introduzir mais propósitos*.

2. A estratégia *Re-frasear/detalhar/especificar o propósito* ou a estratégia *Introduzir mais propósitos* aparece **antes** da estratégia *Indicar o propósito principal*

    - A estratégia *Indicar o propósito principal* deve enunciar o objetivo principal do seu trabalho, por isso deve ser a primeira estratégia de propósito a aparecer no resumo. As estratégias *Re-frasear/detalhar/especificar o propósito* e *Introduzir mais propósitos* são estratégias complementares e devem aparecer depois do propósito principal. Reorganize essas estratégias.

3. Alguma estratégia de *Resultado* aparece **antes** da estratégia *Indicar o propósito principal*

    - Antes de dizer quais foram os resultados do seu trabalho, você deve dizer quais eram seus objetivos. Por isso, a estratégia *Indicar o propósito principal* deve aparecer antes de qualquer uma das estratégias de Resultado. Reorganize sua estrutura para que a estratégia *Indicar o propósito principal* venha antes das estratégias de Resultado.

4. Alguma estratégia de *Metodologia* aparece **antes** da estratégia *Indicar o propósito principal*

    - Antes de dizer os métodos utilizados no seu trabalho é preciso que o leitor saiba quais eram seus objetivos. Por isso, a estratégia *Indicar o propósito principal* deve vir antes das estratégias de Metodologia. Reorganize sua estrutura para que a estratégia *Indicar o propósito principal* apareça antes das estratégias de Metodologia.

5. Alguma estratégia de *Conclusão* aparece **antes** da estratégia *Indicar o propósito principal*

    - As estratégias de Conclusão servem para dar um "fecho" ao resumo. Por isso, é mais natural que elas apareçam no final. Antes de fazer qualquer conclusão, você precisa dizer ao leitor quais eram os objetivos do seu trabalho. Reorganize sua estrutura de forma que a estratégia *Indicar o propósito principal* apareça antes das estratégias de Conclusão.

6. Alguma estratégia de *Conclusão* aparece **antes** de qualquer estratégia de *Metodologia* ou de *Resultado*

    - As estratégias de *Conclusão* servem para dar um "fecho" ao resumo. Por isso, é mais natural que elas apareçam no final. Antes de fazer alguma conclusão, é melhor dizer quais foram seus métodos e seus resultados. Reorganize sua estrutura para que informações sobre a metodologia ou sobre os resultados apareçam antes das conclusões.

7. A estratégia *Introduzir a pesquisa a partir da grande área* aparece **mais de uma vez**

    - A estratégia *Introduzir a pesquisa a partir da grande área* fornece uma contextualização bastante abrangente e dispensa sua repetição. Portanto, é melhor que você deixe apenas uma ocorrência dessa estratégia em sua estrutura.

8. A estratégia *Introduzir a pesquisa a partir da grande área* aparece **depois** da estratégia *Indicar o propósito principal*

    - A estratégia *Introduzir a pesquisa a partir da grande área* fornece uma contextualização bastante abrangente e tem por objetivo guiar o leitor da grande área de pesquisa para a sua área de trabalho mais específica. Essa estratégia prepara o leitor para entender melhor o propósito do seu trabalho. Reorganize sua estrutura para que a estratégia *Introduzir a pesquisa a partir da grande área* apareça antes da estratégia *Indicar o propósito principal*

9. A estratégia *Comentar/discutir (generalizar/explicar/comparar) resultados* aparece **antes** de *Apresentar resultados* e *Descrever o artefato* (*Comentar...* aparece antes das outras estratégias de *Resultado*)

    - Antes de discutir seus resultados você deve apresentá-los ao seu leitor. Por isso, a estratégia *Comentar/discutir (generalizar/explicar/comparar) resultados* deve aparecer depois da estratégia *Apresentar resultados* ou então depois da estratégia *Descrever o artefato*. Reorganize suas estratégias de Resultado.

10. Alguma estratégia de *Contexto*, que não a de *Familiarização*, aparece **depois** da estratégia *Indicar propósito principal* e **sem que preceda** outra estratégia de propósito.

    - As estratégias de *Contexto* servem para familiarizar o leitor com o leitor com sua área de pesquisa e para ressaltar sua relevância. Em geral, essas estratégias aparecem no inicio do resumo e precedem a estratégia *Indicar o propósito principal*. Em sua estrutura existe uma estratégia de *Contexto* após a estratégia *Indicar o propósito principal*. Reorganize sua estrutura para que toda contextualização seja feita antes da indicação do propósito principal.

11. Alguma estratégia de *Lacuna* aparece **depois** da estratégia *Indicar* o *propósito principal* e **sem que preceda** outra estratégia de propósito.

    - As estratégias de *Lacuna* servem para indicar alguma lacuna numa área de pesquisa que vale a pena ser investigada. Dessa forma, o mais comum é que elas apareçam juntamente com a contextualização e que precedam a estratégia *Indicar o propósito principal*. Em sua estrutura existe uma ou mais estratégias de *Lacuna* após a estratégia *Indicar o propósito principal*. Reorganize sua estrutura para que as lacunas indicadas precedam o propósito principal.

3ª Classe de regras – Sugestões de conteúdo

Mensagem padrão: ***"Enriqueça sua estrutura!"***

Mensagens específicas:

1. Faltam estratégias de *Contexto* e *Lacuna*

    - Que tal acrescentar *Contexto* e *Lacuna*? As estratégias de contextualização ajudam o leitor a entender melhor o escopo do seu trabalho. Com essas estratégias você pode convencer o leitor de que sua área de pesquisa é relevante e merece ser explorada. Vale a pena acrescentar alguma estratégia de *Contexto*! A contextualização também prepara o leitor para compreender melhor o problema que você pretende atacar em sua pesquisa. Esse problema é enunciado através das estratégias de *Lacuna*. A declaração de uma lacuna em sua área de pesquisa complementa a contextualização e motiva o propósito do seu trabalho. Escolhas as estratégias mais adequadas ao seu resumo.

2. Faltam estratégias de *Contexto*

    - Você colocou uma ou mais estratégias de *Lacuna* na sua estrutura e isso é bom, pois apontando uma ou mais lacunas numa área de pesquisa você justifica os propósitos do seu trabalho. No entanto, você não usou nenhuma estratégia de *Contexto*. Apontar uma lacuna a ser preenchida motiva o propósito do seu trabalho, mas contextualizar essa lacuna em uma área de pesquisa também é importante. Por isso, estratégias de *Lacuna* geralmente são acompanhadas de estratégias de *Contexto*. Escolha as estratégias de contexto que sejam mais adequadas ao seu resumo.

3. Faltam estratégias de *Lacuna*

    - Você utilizou uma ou mais estratégias de *Contexto*, o que contribuirá para que o leitor entenda em que área de pesquisa seu trabalho se situa. No entanto, você não usou nenhuma estratégia de *Lacuna*. Indicando uma lacuna numa linha de pesquisa, você pode justificar seus objetivos e convencer o leitor da relevância do seu trabalho. Por isso, é comum que estratégias de *Lacuna* acompanhem estratégias de *Contexto*. Escolha uma das estratégias de *Lacuna* que seja mais adequada ao seu resumo.

4. Faltam estratégias de *Conclusão*

    - Você não utilizou nenhuma estratégia de *Conclusão* em sua estrutura. As estratégias de *Conclusão* podem ser utilizadas para dar um "fecho" ao resumo e, dessa forma, torná-lo um texto mais completo. Em um resumo de dissertação/tese, é especialmente bom destacar as contribuições do trabalho. Para isso, acrescente a estratégia *Apresentar contribuições/valor do trabalho*.

4ª Classe de regras – Sugestões de ordem

Mensagem padrão: ***"Melhore a organização da sua estrutura!"***

Mensagens específicas:

1. Alguma estratégia de *Lacuna* aparece **depois** da estratégia *Indicar o propósito principal* e **antes** da estratégia de *Introduzir mais propósitos.*

    - As estratégias de *Lacuna* servem para indicar alguma lacuna numa área de pesquisa que vale a pena ser investigada. Dessa forma, o mais comum é que elas apareçam juntamente com a contextualização e que precedam a estratégia *Indicar o propósito principal*. Em sua estrutura existe uma ou mais estratégias de *Lacuna* após a estratégia *Indicar o propósito principal*. Você pode estar utilizando essa lacuna para introduzir a estratégia *Introduzir o mais propósito.* Tome cuidado. A intercalação dessas estratégias pode confundir o leitor em relação ao seu propósito principal. Utilize **marcadores** que reforcem a distinção entre seus propósitos principais e seus propósitos secundários.

2. Alguma estratégia de *Contexto*, que não a de *Familiarização*, aparece **depois** da estratégia *Indicar o propósito principal* e **antes** da estratégia *Introduzir mais propósitos*

    - As estratégias de *Contexto* servem para familiarizar o leitor com o leitor com sua área de pesquisa e para ressaltar sua relevância. Em geral, essas estratégias aparecem no inicio do resumo e precedem a estratégia *Indicar o propósito principal*. Em sua estrutura existe uma estratégia de *Contexto* após a estratégia *Indicar o propósito principal*. Você pode estar utilizando essa nova contextualização para introduzir a estratégia *Introduzir mais propósitos*. Tome cuidado. A retomada da contextualização pode confundir o leitor em relação ao seu propósito principal. Utilize **marcadores** que reforcem a distinção entre seus propósitos principais e seus propósitos secundários.

Anexo 2 – Predicados Prolog de Busca

Aqui estão reproduzidos os predicados Prolog utilizados na recuperação de casos similares, os quais estão contidos no arquivo **scipo.pro**. Eles não estão necessariamente na mesma ordem, pois existem outros predicados do projeto AMADEUS que atualmente não estão sendo utilizados e que foram mantidos devido ao caráter experimental da implementação.

- **Predicados auxiliares:**

```
- Comprimento de uma lista
tamanho([],0).
tamanho([X|C],N) :- tamanho(C,N1), N is N1 + 1.

- Concatena duas listas
copia([],L,L).
copia([X],L,[X|L]).
copia([X|R],L,[X|LN]):- copia(R,L,LN).

- Elimina o valor de similaridade de uma lista de casos recuperados
limpa_vs([],[]).
limpa_vs([(X,VS)|Resto1],[X|Resto2]) :- limpa_vs(Resto1,Resto2).

- Ordena uma lista de casos recuperados em ordem decrescente de valor de similaridade
ordena(L,L_saida) :- quicksort(L,L_saida).

quicksort([],[]).
quicksort([(Caso,X)|C],L_ord) :-
        particionar((Caso,X),C,Men,Mai),
        quicksort(Men,Menord),
        quicksort(Mai,Maiord),
        concatenar(Menord,[(Caso,X)|Maiord],L_ord).

particionar(_,[],[],[]).
particionar((Caso1,X),[(Caso2,Y)|C],[(Caso2,Y)|Men],Mai) :-
        X < Y, !,
        particionar((Caso1,X),C,Men,Mai).
particionar((Caso1,X),[(Caso2,Y)|C],Men,[(Caso2,Y)|Mai]) :-
        particionar((Caso1,X),C,Men,Mai).

concatenar([],L,L).
```

```
concatenar([E|L1],L2,[E|L3]) :-
        concatenar(L1,L2,L3).

- Lê o arquivo features.txt e retorna a lista de componentes estruturais e estrategias encontradas
leitura(L) :-
        open('features.txt', 'read', Features),
        processa_arq([],Features,NL1),
        inverte(NL1,L),
        close(Features).

processa_arq(L1,Features,L) :-
        read(Features,Comp),
        processa(L1,Features,Comp,L).

processa(L1,Features,end,L1) :- !.
processa(L1,Features,Comp,L) :-
        read(Features,Est),
        monta(Comp,Est,Elem),
        insere(Elem,L1,NL1),
        processa_arq(NL1,Features,L).

insere(E,L,[E|L]).

inverte(L,LI) :- inv1(L,[],LI).

inv1([],L,L).
inv1([E|C],L,LI) :- inv1(C,[E|L],LI).

monta(Comp,Est,c(Comp,s(Est,_))).

- Exclui casos com valor de similaridade abaixo de 'Lim'
retira_menores([(X,VS)|Cauda], Lim, [(X,VS)|Cauda1]) :-
        VS >= Lim,
        !,
        retira_menores(Cauda, Lim, Cauda1).
retira_menores(Lista_ord, Lim, []).
```

- **Requisição igual ao caso (comparação entre listas):**

```
iguais(L1,X) :- case(X,L2,_),
        listas_iguais(L1,L2).

- Teste de igualdade entre listas
listas_iguais([],[]).
listas_iguais([c(X,s(X1,_))|R1],[c(X,s(X1,_))|R2]) :- listas_iguais(R1,R2).
```

- **Requisição totalmente contida no caso (comparação entre listas):**

```
similar1(L1,(X,Valor_Sim)):- case(X,L2,_),
        \+ listas_iguais(L1,L2), sublista(L2,L1),
        tamanho(L1,T1), tamanho(L2,T2),
        Valor_Sim is 100 * (T1/T2).
```

- Teste de inclusão entre listas

```
sublista(Lista,Sublista) :- prefixo(Sublista,Lista).
sublista([_|R],SL) :- sublista(R,SL).

prefixo([],_).
prefixo([c(X,s(X1,_))|RestoSL],[c(X,s(X1,_))|RestoL]) :-
        prefixo(RestoSL,RestoL).
```

- **Requisição parcialmente contida no caso (comparação entre listas):**

```
similar2(L1,(X,Valor_Sim)) :- case(X,L2,_),
        \+ listas_iguais(L1,L2),\+ sublista(L2,L1),
        inter_listas(L1,L2,L),L \== [],tamanho(L,TL),
        tamanho(L1,T1),tamanho(L2,T2), T1 =< T2,
        Valor_Sim is 100 * (TL/(T1 + T2 - TL)).
```

- Teste de intersecção entre listas

```
inter_listas(L1,L2,ML) :-
        findall(L,(sublista(L1,L), sublista(L2,L)),M),
        maior_lista(M,ML).

maior_lista([],[]).
maior_lista([L],L).
maior_lista([L1|Cauda],L) :- maior_lista(Cauda,MLC), !,
        tamanho(L1,TL1),tamanho(MLC,TMLC), testa_maior_l(TL1,TMLC,L1,MLC,L).

testa_maior_l(TL1,TMLC,L1,MLC,L) :- TL1 > TMLC, !,L = L1.
testa_maior_l(TL1,TMLC,L1,MLC,L) :- L = MLC.
```

- **Caso totalmente contido na requisição (comparação entre listas):**

```
similar3(L1,(X,Valor_Sim)):- case(X,L2,_),
        \+ listas_iguais(L1,L2), sublista(L1,L2),
        tamanho(L1,T1), tamanho(L2,T2),
        Valor_Sim is 100 * (T2/T1).
```

- **Caso parcialmente contido na requisição (comparação entre listas):**

```
similar4(L1,(X,Valor_Sim)) :- case(X,L2,_),
        \+ listas_iguais(L1,L2), \+ sublista(L1,L2),
        inter_listas(L1,L2,L), L \== [], tamanho(L,TL),
        tamanho(L1,T1),tamanho(L2,T2), T2 < T1,
        Valor_Sim is 100 * (TL/(T1 + T2 - TL)).
```

- **Requisição igual ao caso (comparação entre conjuntos):**

```
iguais_semord(L1, X) :-
        case(X, L2, _),
        \+ listas_iguais(L1,L2),
        retirar_rep(L1, L11),
        retirar_rep(L2, L22),
        conj_iguais(L11, L22).

- Retira os elementos repetidos de uma lista
retirar_rep([], []).
retirar_rep([c(X,s(X1,_))|Cauda], [c(X,s(X1,_))|Cauda1]) :-
        retirar_todas(c(X,s(X1,_)), Cauda, Lista),
        retirar_rep(Lista, Cauda1).

retirar_todas(_, [], []).
retirar_todas(c(X,s(X1,_)), [c(X,s(X1,_))|Cauda], L) :-
        retirar_todas(c(X,s(X1,_)), Cauda, L).
retirar_todas(c(X,s(X1,_)), [c(Y,s(Y1,_))|Cauda], [c(Y,s(Y1,_))|Cauda1]) :-
        \+ igualdade(c(X,s(X1,_)), c(Y,s(Y1,_))),
        retirar_todas(c(X,s(X1,_)), Cauda, Cauda1).

- Testa igualdade entre pares formados por componentes e estratégias
igualdade(c(X,s(X1,_)), c(Y,s(Y1,_))) :- X == Y, X1 == Y1.

- Igualdade entre conjuntos
conj_iguais([], []).
conj_iguais([c(X,s(X1,_))|L1], L2) :-
        remover(c(X,s(X1,_)), L2, L3),
        conj_iguais(L1, L3).

remover(c(X,s(X1,_)), [c(X,s(X1,_))|Y], Y) :- !.
remover(c(X,s(X1,_)), [c(Y,s(Y1,_))|L1], [c(Y,s(Y1,_))|L2]) :-
        remover(c(X,s(X1,_)), L1, L2).
```

- **Requisição totalmente contida no caso (comparação entre conjuntos):**

```
similar1_semord(L1, (X,Valor_Sim)) :-
        case(X, L2, _),
        \+ listas_iguais(L1, L2), \+ sublista(L2, L1),
        retirar_rep(L1, L11),
        retirar_rep(L2, L22),
        \+ conj_iguais(L11, L22), subconjunto(L22, L11),
        tamanho(L11, T1), tamanho(L22, T2),
        Valor_Sim is 100 * (T1/T2).
```

- Teste de inclusão entre conjuntos

```
subconjunto([], []).
subconjunto(Conj, Subconj) :-
        intersec_conj(Conj, Subconj, Inter),
        conj_iguais(Subconj, Inter).
```

- Teste de intersecção entre conjuntos

```
intersec_conj([], _, []).
intersec_conj([c(X,s(X1,_))|Cauda], L, [c(X,s(X1,_))|U]) :-
        pertence(c(X,s(X1,_)), L),
        !,
        intersec_conj(Cauda, L, U).
intersec_conj([_|Cauda], L, U) :-
        intersec_conj(Cauda, L, U).
```

- Testa se um elemento pertence a um conjunto

```
pertence(c(X,s(X1,_)), [c(X,s(X1,_))|_]).
pertence(c(X,s(X1,_)), [_|Cauda]) :-
        pertence(c(X,s(X1,_)),Cauda).
```

- **Requisição parcialmente contida no caso (comparação entre conjuntos):**

```
similar2_semord(L1, (X,Valor_Sim)) :-
        case(X, L2, _),
        \+ listas_iguais(L1, L2),
        \+ sublista(L2, L1),
        inter_listas(L1, L2, Temp),
        retirar_rep(Temp, LT),
        retirar_rep(L1, L11),
        retirar_rep(L2, L22),
```

```
        \+ conj_iguais(L11, L22), \+ subconjunto(L22, L11),
        intersec_conj(L11, L22, L),
        tamanho(LT, TLT), tamanho(L, TL), TL > TLT,
        tamanho(L11, T1), tamanho(L22, T2), T1 =< T2,
        Valor_Sim is 100 * (TL / (T1 + T2 - TL)).
```

- **Caso totalmente contido na requisição (comparação entre conjuntos):**

```
similar3_semord(L1, (X,Valor_Sim)) :-
        case(X, L2, _),
        \+ listas_iguais(L1, L2), \+ sublista(L1, L2),
        retirar_rep(L1, L11),
        retirar_rep(L2, L22),
        \+ conj_iguais(L11, L22), subconjunto(L11, L22),
        tamanho(L11, T1), tamanho(L22, T2),
        Valor_Sim is 100 * (T2/T1).
```

- **Caso parcialmente contido na requisição (comparação entre conjuntos):**

```
similar4_semord(L1, (X,Valor_Sim)) :-
        case(X, L2, _),
        \+ listas_iguais(L1, L2),
        \+ sublista(L1, L2),
        inter_listas(L1, L2, Temp),
        retirar_rep(Temp, LT),
        retirar_rep(L1, L11),
        retirar_rep(L2, L22),
        \+ conj_iguais(L11, L22), \+ subconjunto(L11, L22),
        intersec_conj(L11, L22, L),
        tamanho(LT, TLT), tamanho(L, TL), TL > TLT,
        tamanho(L11, T1), tamanho(L22, T2), T2 < T1,
        Valor_Sim is 100 * (TL / (T1 + T2 - TL)).
```

- **1º Nível de recuperação de casos:**

```
todas_estrat(L) :-
        consult('Case_base_resumos.pro'),
        leitura(Req),
        findall((X,100), iguais(Req,X), L_I),
        findall((X,Valor_Sim), similar1(Req, (X,Valor_Sim)), L11),
        ordena(L11, L_1),
```

```
        copia(L_I, L_1, L).
```

- **2º Nível de recuperação de casos:**

```
algumas_estrat(L) :-
        consult('Case_base_resumos.pro'),
        leitura(Req),
        findall((X,Valor_Sim), similar3(Req, (X,Valor_Sim)), L33),
        findall((X,Valor_Sim), similar2(Req, (X,Valor_Sim)), L22),
        findall((X,Valor_Sim), similar4(Req, (X,Valor_Sim)), L44),
        ordena(L33, L_3o),
        ordena(L22, L_2o),
        ordena(L44, L_4o),
        retira_menores(L_3o, 25, L_3),
        retira_menores(L_2o, 25, L_2),
        retira_menores(L_4o, 25, L_4),
        copia(L_2, L_4, L_TEMP),
        copia(L_3, L_TEMP, L).
```

- **3º Nível de recuperação de casos:**

```
todas_estrat_semord(L) :-
        consult('Case_base_resumos.pro'),
        leitura(Req),
        findall((X,100), iguais_semord(Req,X), L_I),
        findall((X,Valor_Sim), similar1_semord(Req, (X,Valor_Sim)), L11),
        ordena(L11, L_1),
        copia(L_I, L_1, L).
```

- **4º Nível de recuperação de casos:**

```
algumas_estrat_semord(L) :-
        consult('Case_base_resumos.pro'),
        leitura(Req),
        findall((X,Valor_Sim), similar3_semord(Req, (X,Valor_Sim)), L33),
        findall((X,Valor_Sim), similar2_semord(Req, (X,Valor_Sim)), L22),
        findall((X,Valor_Sim), similar4_semord(Req, (X,Valor_Sim)), L44),
        ordena(L33, L_3o),
        ordena(L22, L_2o),
        ordena(L44, L_4o),
        retira_menores(L_3o, 40, L_3),
```

```
        retira_menores(L_2o, 40, L_2),
        retira_menores(L_4o, 40, L_4),
        copia(L_2, L_4, L_TEMP),
        copia(L_3, L_TEMP, L).
```

- **Recuperação do nome de todos os casos da base:**

```
consulta_base(L) :-
        consult('Case_base_resumos.pro'),
        findall(X, case(X, _, _), L).
```

- **Recuperação de casos que contêm determinada estratégia:**

```
consulta_estr(L) :-
        consult('Case_base_resumos.pro'),
        leitura(L1),
        findall((X,Valor_Sim), consulta_estr(L1,(X,Valor_Sim)), L_D),
        limpa_vs(L_D, L).

consulta_estr(L1, (X,Valor_Sim)) :-
        case(X, L2, _),
        retirar_rep(L1, L11),
        retirar_rep(L2, L22),
        subconjunto(L22, L11),
        tamanho(L11, T1), tamanho(L22, T2),
        Valor_Sim is 100 * (T1/T2).
```

Anexo 3 – Lista de Arquivos da Implementação

Case_base_resumos.pro – Estrutura dos casos da base de Resumos codificada em Prolog.

consulta_marcadores.html.inc – Contém a lista de marcadores discursivos. É um arquivo HTML estático.

consulta_marcadores.php – Exibe a lista de marcadores discursivos carregando o arquivo consulta_marcadores.html.inc numa página com cabeçalho, menu e rodapé. É chamado pelo menu principal da aplicação, se o usuário não estiver na fase de redação.

consulta_marcadores_ popup.php – Também exibe a lista de marcadores discursivos, dessa vez carregando o arquivo consulta_marcadores.html.inc numa página simples para exibição em janelas *pop-ups*. É chamado pelo menu principal da aplicação, se o usuário estiver na fase de redação.

debug* – Arquivos usados para visualizar os resultados da recuperação baseada em casos feita em Prolog. Sua implementação foi realizada no início do projeto, e não acompanhou a evolução do restante dos scripts que também fazem consultas Prolog, pois estes apenas são usados para testes.

features.txt – Armazena as estratégias selecionadas pelo usuário. É usado pelos predicados Prolog na recuperação de casos.

footer.html.inc – Rodapé das páginas. Todas as páginas que contém rodapé utilizam esse arquivo.

funcoes.js – Funções JavaScript para manipulação dos elementos da interface. Todas as funções JavaScript criadas, exceto as do sistema de críticas (arquivo resumos_criticas.js) estão localizadas nesse arquivo. Segue abaixo uma lista dessas funções:

- *sobe*: Move para cima o elemento selecionado num objeto HTML do tipo Select. É usada no arquivo resumos_redacao_estr.php.

- *desce*: Move para baixo o elemento selecionado de um objeto Select. É usada no arquivo resumos_redacao_estr.php.

- *copia*: Faz uma cópia do elemento selecionado de um objeto Select para outro. É usada no arquivo resumos_redacao_estr.php.

- *remove*: Apaga o item selecionado de um objeto Select. É usada no arquivo resumos_redacao_estr.php.

- *remove_tudo*: Apaga todos os itens de um objeto Select. É usada no arquivo resumos_redacao_estr.php.

- *agrupa_selecao*: Agrupa todos os elementos de um objeto Select num objeto Input Text, para posterior envio do formulário de seleção de estratégias do arquivo resumos_redacao_estr.php. Isto é necessário pois apenas o item selecionado de um objeto Select é mandado para o servidor.

- *agrupa_selecao2*: Junta os valores de todos os objetos Textarea de um formulário em um único objeto Input Text. Esta função é usada na fase de redação de textos (arquivo resumos_redacao.php) quando o texto já digitado pelo usuário deve ser mantido após a realização de uma ação em outra página que não a página de composição, ou seja, quando o texto digitado vai "percorrer" outras páginas antes de voltar à fase de criação do texto.

- *exibe_caso*: Exibe o caso cujo nome está selecionado numa lista Select. É usado no arquivo resumos_consulta_base.php.inc.

- *exibe_caso2*: Exibe o caso cujo nome é passado como parâmetro. É usado no arquivo resumos_consulta_estr.php.inc.

- *mais_casos*: Apresenta mensagens de confirmação para apresentação de mais casos na fase de recuperação de casos (usada no arquivo resumos_redacao_casos.php.inc).

- *conta_palavras*: Conta o número de palavras num objeto Textarea. Esta implementação, usada no arquivo resumos_redacao.php, define uma palavra como sendo uma seqüência de caracteres alfanuméricos, delimitada por qualquer outro caractere que não seja alfanumérico.

- *atualiza_contadores*: Atualiza os contadores de caracteres e palavras, os quais estão contidos num formulário do arquivo resumos_redacao.php. É usada quando a página de composição é recarregada.

- *mostra_exemplos_estr*: Dada uma lista de estratégias num objeto Select, exibe exemplos da estratégia selecionada nessa lista. Chamada no arquivo resumos_redacao_estr.php.

- *inicia_selecao*: Usado na tela de seleção de estratégias para habilitar os botões quando o usuário inicia a escolha de estratégias. Chamada no arquivo resumos_redacao_estr.php.

- *reinicia_selecao*: Usado também na tela de seleção de estratégias, para desabilitar os botões quando o usuário exclui todas as estratégias que haviam sido selecionadas. Usada no arquivo resumos_redacao_estr.php.

- *junta_para_rtf*: Agrupa num objeto Input Text os textos contidos em todos os objetos Textarea de um formulário. É usado logo antes da geração do arquivo RTF, no arquivo resumos_redacao.php.

gera_rtf.php – Cria um arquivo no formato RTF, através da gravação do arquivo resumo.fo e da execução da ferramenta Jfor.

geral.css – Arquivo CSS que especifica toda a aparência do sistema.

header.html.inc – Cabeçalho das páginas. Todas as páginas que contém cabeçalho utilizam esse arquivo.

index.php – Página inicial do sistema.

jfor-0.7.1.jar – Arquivo da ferramenta Jfor.

jfor.jar – Arquivo da ferramenta Jfor.

menu_inicial.html.inc – Barra de menu da página inicial.

menu_resumos.html.inc – Primeira barra de menu exibida ao escolher o item "Resumos" no menu da página inicial.

menu_resumos1.html.inc – Barra de menu exibida ao escolher o item "Iniciar Redação".

menu_resumos2.html.inc – Barra de menu exibida ao escolher o item "Exemplos de Estratégias", se já não tiver escolhido o item "Iniciar Redação". Caso contrário, os exemplos de estratégias serão exibidos numa nova janela (*pop-up*).

menu_resumos3.html.inc – Barra de menu exibida ao escolher o item "Navegação pela Base", se já não tiver escolhido o item "Iniciar Redação". Caso contrário, os casos da base serão exibidos numa nova janela.

menu_resumos4.html.inc – Barra de menu exibida ao escolher o item "Marcadores Discursivos", se já não tiver escolhido o item "Iniciar Redação". Caso contrário, os marcadores serão exibidos numa nova janela.

mostra_caso.php – Formata os arquivos XML para serem exibidos, utilizando o parser XML do PHP.

phpinfo.html – Contém informações a respeito da instalação PHP utilizada durante o desenvolvimento do sistema. Não é usado diretamente na implementação, mas pode ser útil para futuras consultas.

resumo.fo – Arquivo XSL-FO usado para gerar o arquivo RTF.

resumos/ – Diretório que contém a base de resumos em XML.

resumos_algumas_estrat.sh – Script Shell que realiza a consulta Prolog para execução do 2º Nível de recuperação de casos similares, conforme Seção 2.5.2. Esse script é chamado pelo arquivo resumos_redacao_casos.php.inc.

resumos_algumas_estrat_semord.sh – Script Shell que realiza a consulta Prolog para execução do 4º Nível de recuperação de casos similares, conforme Seção 2.5.2. Esse script é chamado pelo arquivo resumos_redacao_casos.php.inc.

resumos_consulta_base.php – Exibe a página de "Navegação pela Base", carregando o arquivo resumos_consulta_base.php.inc numa página com cabeçalho, menu e rodapé. É chamado pelo menu principal da aplicação, se o usuário não estiver na fase de redação.

resumos_consulta_base.php.inc – Implementa a opção "Navegação pela Base". Utiliza o script resumos_consulta_base.sh para obter os nomes de todos os casos da base e disponibilizá-los para visualização.

resumos_consulta_base_popup.php – Também exibe a página de "Navegação pela Base", dessa vez carregando o arquivo resumos_consulta_base.php.inc numa página simples para

exibição em janelas *pop-ups*. É chamado pelo menu principal da aplicação, se o usuário estiver na fase de redação.

resumos_consulta_base.sh – Script Shell que realiza a consulta Prolog para recuperação dos nomes de todos os casos da base. Esse script é chamado pelo arquivo resumos_consulta_base.php.inc.

resumos_consulta_estr.php – Exibe a página de "Exemplos de Estratégias", carregando o arquivo resumos_consulta_estr.php.inc numa página com cabeçalho, menu e rodapé. É chamado pelo menu principal da aplicação, se o usuário não estiver na fase de redação.

resumos_consulta_estr.php.inc – Implementa a opção "Exemplos de Estratégias". Escreve no arquivo features.txt a estratégia e o respectivo componente escolhido, utiliza o script resumos_consulta_estr.sh para obter os nomes de todos os casos da base que fazem uso dessa estratégia e faz uso do parser XML do PHP para exibir os trechos de Resumo dos casos recuperados.

resumos_consulta_estr_popup.php – Também exibe a página de "Exemplos de Estratégias", dessa vez carregando o arquivo resumos_consulta_estr.php.inc numa página simples para exibição em janelas *pop-ups*. É chamado pelo menu principal da aplicação, se o usuário estiver na fase de redação.

resumos_consulta_estr.sh – Script Shell que realiza a consulta Prolog para recuperação dos nomes de todos os casos da base que contêm uma estratégia em específico. Esse script é chamado pelo arquivo resumos_consulta_estr.php.inc, e também lê a lista de componentes e estratégias do arquivo resumos_lista_estr.html.inc.

resumos_criticas.js – Arquivo JavaScript que aplica as regras de críticas listadas no Anexo 1. É usado nos arquivos resumos_redacao_estr.php e resumos_criticas.php, por meio da função critica_selecao. As funções implementadas nesse arquivo estão listadas abaixo:

- *critica_selecao*: É a única função chamada fora deste arquivo. Executa todas as categorias de críticas e sugestões, escolhe o que deverá ser exibido ao usuário e retorna verdadeiro se não existirem críticas. É possível, através de um parâmetro, especificar se as mensagens serão exibidas ao usuário, ou se apenas um teste será feito.

- *contem* - Verifica se uma determinada estratégia está presente no conjunto de estratégias selecionadas pelo usuário. Retorna o número de ocorrências da estratégia na seleção.

- *antes* - Verifica se determinada estratégia vem antes de outra estratégia no conjunto de estratégias selecionadas.

- *criticas_conteudo* - Verifica as críticas de conteúdo. Retorna verdadeiro se não houver críticas.

- *criticas_ordem* - Verifica as críticas de ordem. Retorna verdadeiro se não houver críticas.

- *sugestoes_conteudo* - Verifica as sugestões de conteúdo. Retorna verdadeiro se não houver sugestões.

- *sugestoes_ordem* – Verifica as sugestões de ordem. Retorna verdadeiro se não houver sugestões.

resumos_criticas.php – Arquivo usado na fase de críticas e sugestões, para exibir as informações que o script resumos_criticas.js fornece, através da função critica_selecao, de acordo com as estratégias escolhidas pelo usuário.

resumos_entrada.php – Página inicial da seção de Resumos.

resumos_lista_estr.html.inc – Contém a lista de componentes e estratégias disponíveis usada nos arquivos resumos_consulta_estr.php.inc e resumos_redacao_estr.php.

resumos_redacao.php – Monta a página de composição de um Resumo.

resumos_redacao_casos.php – Exibe a página de recuperação de casos similares, carregando o arquivo resumos_redacao_casos.php.inc numa página com cabeçalho, menu e rodapé. É chamado pela página de seleção de estratégias, resumos_redacao_estr.php.

resumos_redacao_casos.php.inc – Implementa a parte de recuperação de casos similares da fase de redação. Escreve no arquivo features.txt as estratégias e os respectivos componentes escolhidos, e utiliza os scripts resumos_todas_estrat.sh, resumos_algumas_estrat.sh, resumos_todas_estrat_semord.sh e resumos_algumas_estrat_semord.sh para executar os quatro níveis de recuperação de casos, conforme seção 2.5.2.

resumos_redacao_casos_popup.php – Também exibe a página de recuperação de casos similares, dessa vez carregando o arquivo resumos_redacao_casos.php.inc numa página simples para exibição em janelas *pop-ups*. É chamado pela página de composição do Resumo, denominada resumos_redacao.php.

resumos_redacao_estr.php – É a página inicial da fase de redação, onde o usuário escolhe as estratégias que usará em seu Resumo. A lista de componentes e estratégias é lida do arquivo resumos_lista_estr.html.inc.

resumos_todas_estrat.sh – Script Shell que realiza a consulta Prolog para execução do 1º Nível de recuperação de casos similares, conforme Seção 2.5.2. Esse script é chamado pelo arquivo resumos_redacao_casos.php.inc.

resumos_todas_estrat_semord.sh – Script Shell que realiza a consulta Prolog para execução do 3º Nível de recuperação de casos similares, conforme Seção 2.5.2. Esse script é chamado pelo arquivo resumos_redacao_casos.php.inc.

rtf/ – Diretório em que os arquivos RTF são colocados.

scipo.pro – Arquivo com os predicados Prolog de recuperação de casos similares.

xerces-1.2.3.jar – Arquivo da ferramenta Jfor.

xerces.jar – Arquivo da ferramenta Jfor.