DANIEL FAGUNDES DA SILVA

SOFTWARE AGREGADOR DE NOTÍCIAS COM CLASSIFICADOR BAYESIANO

Monografia apresentada como requisito parcial à obtenção do grau de Bacharel em Ciências da Computação, Curso de Graduação em Ciências da Computação, Departamento de Informática e de Estatística, Centro Tecnológico, Universidade Federal de Santa Catarina.

Orientador: Prof. Dr. José Mazzucco Júnior

FLORIANÓPOLIS
2007

TERMO DE APROVAÇÃO

DANIEL FAGUNDES DA SILVA

SOFTWARE AGREGADOR DE NOTÍCIAS
COM CLASSIFICADOR BAYESIANO

Monografia aprovada como requisito parcial para obtenção do grau de Bacharel no Curso de Graduação em Ciências da Computação, Departamento de Informática e Estatística, Centro Tecnológico da Universidade Federal de Santa Catarina, pela seguinte banca examinadora:

Orientador: Prof. Dr. José Mazzucco Júnior
Departamento de Informática e Estatística, UFSC

Prof. José Francisco Danilo de Guadalupe Correa Fletes
Departamento de Informática e Estatística, UFSC

Prof. Dr. Leandro José Komosinski
Departamento de Informática e Estatística, UFSC

Florianópolis, junho de 2007

Agradeço à minha noiva, Paula Heidy, pelo carinho, apoio e compreensão, por todos estes anos;

Agradeço aos meus irmãos, Érico e Jane, por sempre me apontarem o caminho em momentos difíceis;

Em especial agradeço aos meus pais, Américo e Leir, que dedicaram suas vidas ao crescimento de seus filhos.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE SIGLAS

DTD - Document Type Definition

HTML - Hyper Text Markup Language

IEFT - Internet Engineering Task Force

MCF - Meta Content Framework

RDF - Resource Description Framework

RSS - RDF Site Summary, Rich Site Summary, Really Simple Syndication

URL - Universal Resource Locator

W3C - World Wide Web Consortium

XML - Extensible Markup Language

RESUMO

O presente trabalho descreve o desenvolvimento de um software agregador de noticias, capaz de classificar a ordem em que as notícias são apresentadas ao usuário, de acordo com a análise das palavras-chave presentes em cada notícia. Para tanto, discorre sobre a fundamentação teórica necessária para fazer a análise e extração de conteúdo dos formatos de *feeds* mais comumente usados atualmente na Internet. Uma visão geral da teoria básica de Probabilidade necessária para compreensão do processo de classificação também é apresentada, e um exemplo da aplicação do Teorema de Bayes é fornecido. Por fim, é detalhado o processo de desenvolvimento do software agregador de notícia, desde a etapa necessária de extração de palavras-chave presentes em cada notícia até o classificador bayesiano que as utiliza para determinar o provável grau de interesse do usuário para cada notícia.

Palavras-chave: agregador; notícias; classificador; teorema de bayes; filtro bayesiano.

ABSTRACT

This essay describes the development of a news aggregator, capable to classify the order in which the news articles are presented to the user, accordingly with the analysis of the keywords present in each news article. To accomplish this, it describes about the theoretical background required to make the parse and extraction of the content from the most frequent *feed* formats available on Internet today. It is presented an overview about the basic Probability theory required to comprehend the classification process, and it is given an example of the application of the Bayes' Theorem. By the end, the development process of the news aggregator is detailed, from the process required to extract the keywords present in each news article to the bayesian classifier that utilizes them to determine the probable user's interest degree in each news article.

Keywords: aggregator; news; classifier; bayes' theorem; bayesian filter.

1 INTRODUÇÃO

A Internet atual é uma fonte valiosa de informação. O número de usuários que utilizam-na para buscar notícias vem crescendo cada vez mais. Grandes portais de notícias atualizam seu conteúdo com uma frequência impraticável para as mídias tradicionais, como a mídia impressa. Desta forma, é interessante notar que a própria natureza volátil da informação na Internet e a constante atualização das notícias, consideradas como benefícios da mídia *online*, sejam também a principal causa de um problema cada vez mais comum para quem utiliza este tipo de mídia: a **sobrecarga de informação.**

1.1 DEFINIÇÃO DO PROBLEMA

Com o aumento da oferta de notícias *online*, o usuário comum vê-se frente a frente com um volume cada vez maior de conteúdo. O problema da sobrecarga de informação reside no fato de que o leitor em busca informação por vezes não consegue encontrar a informação que deseja em meio à massa de informações que ele não considera interessante. A utilização de softwares e *websites* agregadores de notícias resolvem uma parte do problema, evitando que o usuário tenha de acessar individualmente os *websites* de sua preferência, em busca de conteúdo atualizado. Porém, mesmo a utilização destes recursos não impede que o usuário receba diariamente dezenas, por vezes **centenas**, de notícias das mais diversas fontes e tenha de percorrer, uma a uma, tais notícias em busca de conteúdo relevante.

1.2 OBJETIVO GERAL

A proposta do trabalho atual é desenvolver um aplicativo agregador de notícias, sob a forma de um *website*, que seja capaz de realizar a classificação das

notícias trazidas da Internet, na ordem da mais relevante para a menos relevante, através da análise das palavras-chave contidas em cada notícia, calculando o grau de possível interesse por parte do usuário em uma determinada notícia, através do uso de um classificador *bayesiano*.

1.3 OBJETIVOS ESPECÍFICOS

Os objetivos específicos do presente trabalho são:

- desenvolver uma biblioteca de classes para buscar notícias *online*;
- criar uma base de dados relacional para armazenar as notícias trazidas da Internet;
- elaborar um algoritmo para extração de palavras-chave das notícias;
- desenvolver um classificador utilizando um filtro bayesiano;
- implementar um agregador de notícias sob a forma de um *website*.

2 DISTRIBUIÇÃO DE CONTEÚDO ONLINE

No início da popularização da Internet, uma quantidade limitada de conteúdo, em sua maior parte estático, estava disponível aos seus usuários, e podia ser acessada através do padrão marcação de texto (HTML) adotado como padrão para distribuição do conteúdo *online* através dos *websites*.

Porém, com o aumento da popularidade da Internet e o conseqüente aumento na variedade e complexidade dos serviços oferecidos *online*, novas tecnologias tornaram-se necessárias para facilitar a distribuição da informação.

A distribuição de conteúdo torna o material de um *website* disponível para uso em outros serviços. O conteúdo distribuído, ou *feed*, pode ser tanto a própria informação, entregue diretamente ao usuário, bem como *metadados* – informações a respeito do conteúdo. A distribuição de conteúdo permite que seus usuários visualizem a informação em múltiplos dispositivos e que seus assinantes sejam notificados de maneiras diferentes a respeito de atualizações ocorridas [HAMMERSLEY].

Enquanto a quantidade de conteúdo estático acessível através de mecanismos de buscas fornece informações úteis em tópicos pré-estabelecidos, a possibilidade de acessar um aparente infinito número de fontes de notícias na Internet forneceu aos seus usuários um novo desafio: classificar e filtrar conteúdo interessante de uma maneira eficiente e em tempo hábil.

Devido ao fato de um padrão ser necessário para tornar o conteúdo acessível para uma comunidade global, a tecnologia para distribuir tal conteúdo foi desenvolvida quase da mesma maneira que o conteúdo em si: através do esforço coletivo e colaborativo de pessoas ao redor do mundo, através da Internet.

A tecnologia desenvolvida consiste em formatos de marcação de texto elaborados especificamente para representar, padronizar e facilitar a distribuição do conteúdo disponível *online*.

2.1 HISTÓRICO

O formato atual mais popular para distribuição de conteúdo é o RSS. Porém, antes do formato RSS ser amplamente aceito, diversos formatos similares já existiam para distribuição de conteúdo, mas nenhum alcançou a popularidade necessária, uma vez que a maioria foi concebida para trabalhar com apenas um único serviço *online*.

As origens obscuras das versões atuais do RSS começaram no ano de 1995, com o trabalho Ramanathan V. Guha. Ele desenvolveu uma especificação chamada de *Meta Content Framework* (MCF). Enraizado em trabalhos de sistemas de representação de conhecimento tais como CycL, KRL, e KIF, o objetivo do MCF era descrever objetos, seus atributos e as relações entre eles [HAMMERSLEY].

Na época Guha trabalhava para a Apple junto com outros pesquisadores no *Advanced Technology Group* no desenvolvimento do MFC. O MFC estava sendo desenvolvido como a base de uma aplicação denominada *Project X* , rebatizada mais tarde como *HotSauce*. O HotSauce permitia aos usuários visualizar e navegar em representações 3D de *websites* que possuíssem a descrição de seu conteúdo no formato MCF.

Apesar de popular, o MFC era um projeto experimental. Em 1997 Steve Jobs retornou à gerencia da Apple e muitas atividades de pesquisa sendo desenvolvidas pela Apple foram interrompidas [HAMMERSLEY]. Quando o desenvolvimento do MFC foi descontinuado, Guha deixou a Apple, indo trabalhar para a Netscape.

Na Netscape, Guha encontrou em contato com Tim Bray, co-autor da especificação XML 1.0 e um dos maiores contribuidores desta tecnologia [BRAY]. Guha então decidiu transformar o MFC em uma aplicação do XML [ANDREESSEN]. Guha e Bray adaptaram o MFC para um formato baseado em XML e enviaram a especificação para o World Wide Web Consortium (W3C) em junho de 1997. [GUHA]

Desta combinação do MFC com XML nasceria mais tarde o formato *Resource Description Framework* (RDF). Segundo o W3C, o formato RDF é “uma linguagem de propósito geral para representar informação na *World Wide Web*” [RDF]. Ele é projetado especificamente para a representação de metadados e as relações entre eles. Em sua forma mais completa, é a base do conceito conhecido atualmente como “Web Semântica”, a visão do W3C da informação da web que os computadores conseguem compreender [HAMMERSLEY].

Ainda no ano de 1997, Dave Winer, pioneiro em sua participação do desenvolvimento e utilização de tecnologias populares na Internet atualmente (tais como *weblogs* e *podcasts*), projetou e anunciou seu próprio formato, denominado s*criptingNews*, para distribuição de conteúdo XML para o uso em seu *weblog Scripting News* [WINER].

2.1.1 O Surgimento do RSS

O formato RSS foi inicialmente desenvolvido pela Netscape, em março de 1999, para o uso no portal *My Nestscape*. Na época, a sigla significava *RDF Site Summary* e esta versão do RSS ficou conhecida como RSS 0.9 [MNN].

Em julho de 1999, atendendo a comentários e sugestões, Dan Libby desenvolveu um protótipo nomeado como RSS 0.91, o qual simplificava a versão anterior e incorporava parte do formato de distribuição de conteúdo adotado por Dave Winer. O nome *RDF Site Summary* foi abandonado, uma vez que todas as referências ao formato RDF foram removidas e o formato passou a se basear no XML. Sua justificativa para isto foi de que os distribuidores de conteúdo desejavam mais um formato para sumarizar *websites* do que um formato de metadados; a estrutura rígida do formato RDF não tornava fácil a leitura e dificultava a criação de arquivos RDF úteis que fossem válidos de acordo com a especificação do modelo de dados do RDF. [LIBBY]

Em abril de 2001, como parte da reestruturação do portal *My Netscape*, a AOL removeu o suporte à distribuição de conteúdo através de *feeds* RSS. O *validador* do formato bem como o arquivo de DTD, para o qual muitos *feeds* RSS 0.91 apontavam, foram removidos durante este processo. Em resposta a demanda, Dan Libby conseguiu a restauração do DTD, mas não do validador. A comunidade reagiu negativamente ao repentino abandono do RSS 0.91, na época o formato que mais amplamente aceito na Internet [KING].

Porém, um grupo de trabalho e uma lista de discussão, RSS-DEV, foram estabelecidos para continuar o desenvolvimento do RSS pela comunidade que o utilizava. Na mesma época, Dave Winer publicou uma versão modificada da especificação do formato RSS 0.91 no *website* UserLand, uma vez que ele já havia adotado o formato RSS 0.91 em detrimento do formato que ele havia criado anteriormente, o s*criptingNews*. Ele alegou que a especificação do formato RSS 0.91 era propriedade de sua companhia, a UserLand Software [WINER(2)].

Uma vez que nem o grupo RSS-DEV nem a UserLand Software haviam feito um pedido oficial solicitando o nome do formato, discussões e brigas surgiram sempre que um dos lados alegava que o formato RSS lhes pertencia, o que com o passar do tempo acabou acarretando na divisão no formato.

O grupo RSS-DEV publicou a especificação RSS 1.0 em dezembro de 2000 [RSS-DEV]. Assim como o formato RSS 0.9, esta nova versão era baseada no RDF, porém era mais modular, com muitas das propriedades vindo de vocabulários padrão de metadados, como o *Dublin Core*.

Semanas após o lançamento do formato RSS 1.0, Dave Winer lançou ele mesmo o formato RSS 0.92, com mudanças pequenas e supostamente compatíveis com o formato RSS 0.91 [WINER(3)]. A versão 0.92 foi seguida de rascunhos das versões 0.93 e 0.94, que não se tornaram tão populares quantos as versões até então existentes. Porém, em setembro de 2002, Winer lançou a versão final para o seu sucessor do formato RSS 0.92; esta versão ficou conhecida como RSS 2.0. Com esta

versão, ele enfatizou a mudança de nome para *"Really Simple Syndication"* para denominar a sigla RSS. Desde então, diversas versões do RSS 2.0 têm sido publicadas, mas a numeração da versão do modelo de documento da especificação tem se mantido a mesma.

Em novembro de 2002, o *website* do jornal *The New York Times* começou a oferecer aos seus leitores a possibilidade de assinarem *feeds* de notícias RSS em vários de seus tópicos publicados. Em janeiro de 2003, a adoção pelo *The New York Times* tornou-se um marco para tornar a utilização do RSS como o formato padrão de distribuição de conteúdo.

2.1.2 O Desenvolvimento do Atom

Em 2003, a principal tecnologia utilizada para distribuição *online* de conteúdo era o formato RSS. Membros da comunidade que consideravam o formato RSS 2.0 ainda deficiente não eram capazes de realizar melhorias uma vez que ele havia sido patenteado pela Universidade de Harvard e sua especificação mantida inalterada, de acordo com os termos de uso e licença do formato.

Em junho de 2003, Sam Ruby montou um *wiki* [RUBY] para discutir os aspectos que compunham um registro log bem formado. Em pouco tempo, várias pessoas juntaram-se ao *wiki* para discutir um novo formato de distribuição de conteúdo, levando em consideração os pontos fracos do formato RSS. Não tardou para que um roteiro de desenvolvimento fosse estabelecido e o esforço atraiu mais de 150 participantes, incluindo contribuidores de sites de serviços famosos na Internet como o *Technorati*, *Six Apart*, *LiveJournal*, *Blogger*, *Yahoo!* e *O'Reilly Network*.

Em julho de 2003 a discussão foi movida do *wiki* para uma lista de discussão própria. Em dezembro do mesmo ano uma versão preliminar do formato Atom (versão 0.3) foi liberada, e ganhou uma adoção ampla em ferramentas de distribuição de

conteúdo, em particular em serviços oferecidos pelo Google, como o *Blogger*, *Google News* e *Gmail*.

Em 2004, iniciaram-se discussões a respeito da mudança do projeto Atom para uma entidade padrão, tal com o W3C ou o *Internet Engineering Task Force* – IEFT. O grupo acabou optando pelo IEFT e em junho de 2004 um grupo de trabalho denominado *Atompub* foi formado, dando ao projeto um cronograma de desenvolvimento [IEFT]. É interessante notar que o grupo de trabalho do *Atompub* tem como co-presidente Tim Bray, co-autor da especificação XML.

O rascunho da versão final do formato Atom 1.0 foi publicado em julho de 2005 e foi aceito pelo IEFT como um padrão sugerido para o *"Internet Official Protocol Standards"* em dezembro de 2005 [IETF(2)]. O trabalho de desenvolvimento da especificação tem prosseguido, para o protocolo de publicação de conteúdo e várias extensões para o formato de distribuição.

3 PROBABILIDADE

A teoria do Cálculo das Probabilidades é um ramo da Matemática que está ligado aos estudos de fenômenos que envolvem a aleatoriedade e incerteza, nos quais os resultados variam de uma observação para outra, mesmo que se mantenham as mesmas condições entre as realizações de um experimento [BASTOS]. Ela fornece métodos que permitem quantificar a possibilidade de acontecimentos relacionados aos diversos resultados possíveis [SILVA].

3.1 EXPERIMENTO ALEATÓRIO

Todo experimento que, repetido em condições idênticas, apresenta diferentes resultados, recebe o nome de *experimento aleatório* ou *não-determinístico* [BASTOS]. A variabilidade dos resultados portanto, deve-se ao acaso. Entretanto, se a experiência for repetida um grande número de vezes, pode-se construir um modelo probabilístico e tomar decisões referentes ao processo experimental apenas pelas suas características, sem a necessidade de refazer o experimento [SILVA].

A prática indica que muitos experimentos podem ser realizados como se ocorressem de maneira idêntica. Em tais circunstancias, normalmente é possível construir um modelo matemático satisfatório e empregá-lo no estudo de propriedades e na obtenção de conclusões acerca do experimento; O modelo matemático obtido geralmente é capaz de possibilitar previsões sobre a frequência dos resultados que se espera ocorrerem quando a experiência for repetida. Neste caso, este modelo, chamado de *probabilístico*, funciona como um instrumento matemático que prevê a probabilidade de um possível resultado ocorrer sem que para isso seja necessário repetir o experimento [SILVA].

3.2 ESPAÇO AMOSTRAL

Define-se como *espaço das possibilidades do experimento* ou simplesmente ***espaço amostral*** o conjunto de todos os resultados possíveis de um experimento ε não-determinístico [MEYER]. Comumente o conjunto do espaço amostral é representado pela letra *S*. Cada um dos resultados possíveis de um experimento é um elemento de um espaço amostral *S* [SILVA].

Um mesmo experimento ε (como o lançamento de uma moeda quatro vezes, por exemplo) pode ter espaços amostrais diferentes de acordo com o que se está observando ou mensurando. Desta forma, referimo-nos a *um* espaço amostral e não a *o* espaço amostral do experimento.

3.3 EVENTO

Um evento *A*, relativo à um particular espaço amostral *S*, é um conjunto de resultados de um experimento ε. Na terminologia dos conjuntos, um evento é um *subconjunto* de um espaço amostral *S* [MEYER].

Os eventos podem ser *simples* ou *compostos*. Evento simples é aquele associado a apenas *um* resultado presente em um espaço amostral. Evento composto, por sua vez, é aquele associado à mais de um resultado (ou elemento) presente em um espaço amostral *S*.

Evento certo é aquele que ocorre em qualquer realização do experimento não-determinístico.

Evento impossível é aquele que não ocorre em nenhuma realização de um experimento não-determinístico.

Evento complementar – para um evento *A* qualquer , um evento *A'* é chamado de *evento complementar de A* quando ele é formado por elementos que pertencem ao espaço amostral *S* e não pertencem ao evento *A*.

3.4 FREQUÊNCIA RELATIVA

Uma das características fundamentais do conceito de *experimento não-determinístico* baseia-se no fato de que não é possível saber qual resultado particular ocorrerá a partir de uma realização deste experimento. Em outras palavras, se A for um evento associado à um experimento não-determinístico ε , então não é possível afirmar com certeza que A irá ocorrer ou não. Por esta razão, torna-se importante tentar associar um valor ao evento A que indique a possibilidade de que este evento venha a ocorrer.

Suponha um experimento ε, repetido n vezes, com A sendo um evento associado à este experimento. Considerando n_A o número de vezes que o evento A ocorreu nas n repetições, a razão n_A / n é denominada *frequência relativa* do evento A durante as n repetições do experimento ε.

3.5 PRINCÍPIO DA EQUIPROBABILIDADE

No dia-a-dia, o termo *provável* refere-se à grandeza da porcentagem do que é favorável ao que se deseja em relação a *todos* os resultados. Quando não se tem qualquer informação prévia e as características de um experimento sugerem n resultados possíveis, e todos com igual probabilidade de ocorrência podem-se enumerar os resultados possíveis e descrevê-los como *igualmente prováveis* (equiprováveis).

Desta forma, pode-se dizer que sempre que uma experiência consiste em n resultados possíveis e igualmente prováveis, como por exemplo no lançamento de um dado “honesto” onde cada uma das faces tem a mesma probabilidade de aparecer, a probabilidade de cada resultado ocorrer é $1/n$; A probabilidade assim definida é chamada de *probabilidade a priori*.

Estas considerações resultam no denominado *conceito clássico de probabilidade*, o resultado da divisão entre o número de casos favoráveis e o número de casos possíveis de um experimento [SILVA]:

$$P(A) = \frac{número\ de\ resultados\ favoráveis\ de\ A}{número\ total\ de\ resultados\ possíveis\ de\ A}$$

Estas possibilidades são quantificadas por meio da associação do resultado com um número no intervalo fechado entre 0 e 1, onde números altos indicam que o resultado é mais passível de acontecer. O 0 (zero) indica um resultado que nunca ocorrerá (evento impossível) e o 1 (um) indica que ele, com certeza, ocorrerá (evento certo). Essa idéia é fruto da experiência passada, da observação dos fatos [SILVA].

3.6 PROBABILIDADE CONDICIONAL

Sejam A e B dois eventos de um espaço amostral S, associados a um experimento ε, onde $P(A) > 0$. A probabilidade do evento B ocorrer condicionada ao fato do evento A ter ocorrido, é representada por $P(B|A)$ e interpretada como a *probabilidade de B dado A* ou *probabilidade de B condicionada a A*.

Sempre que for calculado $P(B|A)$, essencialmente é feito o cálculo de $P(B)$ em relação ao *espaço amostral reduzido de A*, em lugar de fazê-lo em relação ao espaço amostral original S. Calcular $P(B)$ equivale a determinar o quão provável será estar no subconjunto do evento B, sabendo que também se deve estar em S. E quando se calcula $P(B|A)$, equivale a determinar o quão provável será estar no subconjunto do evento B sabendo que também se deve estar em A, ou seja, o espaço amostral reduz-se de S para A [MEYER].

Através do conceito de frequência relativa, suponha que um experimento ε tenha sido repetido n vezes. Considerando n_A, n_B e $n_{A \cap B}$ o número de vezes que,

respectivamente, os eventos A, B e $A \cap B$ tenham ocorrido em n repetições (onde $A \cap B$ é o evento composto formado pela ocorrência simultânea dos eventos A e B), a frequência relativa de B nos resultados em que A também tenha ocorrido é representada por:

$$f_B = \frac{n_{A \cap B}}{n_A} = \frac{n_{A \cap B}/n}{n_A/n} = \frac{f_{A \cap B}}{f_A},$$

onde $f_{A \cap B}$ e f_A são as freqüências relativas dos eventos $A \cap B$ e A, respectivamente.

Como mencionado anteriormente, se o número de repetições do experimento for grande, $f_{A \cap B}$ será próxima de $P(A \cap B)$ e f_A será próxima de $P(A)$. Em conseqüência disto, a relação acima sugere que $n_{A \cap B} / n_A$ será próxima de $P(B|A)$. Com isto, é possível estabelecer a seguinte definição:

$$P(B \mid A) = \frac{P(B \cap A)}{P(A)}, \text{ desde que } P(A) > 0.$$

3.7 TEOREMA DA MULTIPLICAÇÃO

A partir do conceito de probabilidade condicional é possível estabelecer facilmente a expressão para o cálculo da probabilidade de dois eventos A e B ocorrerem simultaneamente, conhecida como *Teorema da multiplicação* ou *Teorema do produto*:

$$P(A \cap B) = P(B \mid A) \cdot P(A) \text{ ou } P(A \cap B) = P(A \mid B) \cdot P(B)$$

3.8 EVENTOS INDEPENDENTES

Sejam A e B dois eventos de um espaço amostral S. Os eventos A e B são ditos *independentes* se a probabilidade de um deles ocorrer não afetar a probabilidade de o outro ocorrer, isto é, quando o conhecimento de que o evento B ocorreu não fornece qualquer informação sobre uma possível ocorrência de A.

É possível então verificar que a probabilidade condicionada de um evento A ocorrer dado um evento B é igual à probabilidade absoluta do mesmo evento A ocorrer independentemente de B, ou seja $P(A \mid B) = P(A)$. A mesma afirmação é possível para o evento B, com $P(B \mid A) = P(B)$. Com isto, e considerando o Teorema da multiplicação, é possível estabelecer que:

$$P(A \cap B) = P(A \mid B) \cdot P(B) = P(A) \cdot P(B) \text{ e}$$
$$P(A \cap B) = P(B \mid A) \cdot P(A) = P(B) \cdot P(A)$$

Assim, dois eventos A e B serão considerados independentes se e somente se:

$$P(A \cap B) = P(A) \cdot P(B)$$

3.9 TEOREMA DE BAYES

Sejam A e B dois eventos de um espaço amostral S. Considerando que a probabilidade de observar simultaneamente os eventos A e B , através do Teorema da multiplicação, é dada por $P(A \cap B) = P(B \mid A) \cdot P(A)$ ou $P(A \cap B) = P(A \mid B) \cdot P(B)$, pode-se afirmar que:

$$P(B \mid A) \cdot P(A) = P(A \mid B) \cdot P(B)$$

Rearranjando a expressão pode-se obter:

$$P(B \mid A) = \frac{P(A \mid B) \cdot P(B)}{P(A)}$$

A expressão acima é conhecida como *Teorema de Bayes*, também denominada de *Teorema da Probabilidade das "Causas"*. Ela permite *atualizar* uma crença na possibilidade de ocorrência de um evento *B* (probabilidade *a posteriori*) baseado na experiência prévia conhecida para a possibilidade de o evento *B* ocorrer (probabilidade *a priori*) em conjunto com a observação da ocorrência de uma evidência *A* a respeito do evento *B* [YUDKOWSKY].

Como geralmente não se conhece *P*(*A*), pode-se reescrever esta probabilidade sob a forma:

$P(A) = P(A \cap B) + P(A \cap B')$, onde *B'* é o evento complementar de *B*.

Através do Teorema da multiplicação, pode-se substituir $P(A \cap B)$ e $P(A \cap B')$ na expressão acima e obter:

$$P(A) = P(A \mid B) \cdot P(B) + P(A \mid B') \cdot P(B')$$

Substituindo então no Teorema de Bayes a expressão para *P*(*A*) estabelecida acima, obtém-se a formulação alternativa para o Teorema de Bayes:

$$P(B \mid A) = \frac{P(A \mid B) \cdot P(B)}{P(A \mid B) \cdot P(B) + P(A \mid B') \cdot P(B')}$$

Por fim, considerando os eventos B_1, B_2, ..., B_n como sendo mutuamente exclusivos, formando uma partição do espaço amostral *S*, é possível elaborar o Teorema de Bayes sob a forma:

$$P(B_i \mid A) = \frac{P(A \mid B_i) \cdot P(B_i)}{P(A \mid B_1) \cdot P(B_1) + P(A \mid B_2) \cdot P(B_2) + \ldots + P(A \mid B_n) \cdot P(B_n)}$$

A expressão acima define que desde que os B_i constituam uma partição do espaço amostral, um e somente um dos eventos B_i ocorrerá. Portanto, a expressão acima nos dá a probabilidade de um evento particular B_i ter ocorrido (ou seja, uma "causa") dado que o evento A ocorreu [MEYER].

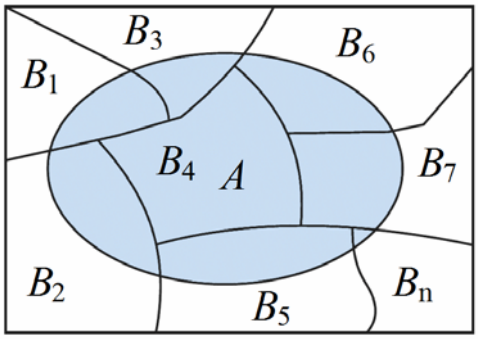

Figura 1. Eventos B_i formando uma partição do espaço amostral S.

4 METODOLOGIA

Nesta seção será apresentada a metodologia utilizada no processo de desenvolvimento da estrutura necessária e da aplicação proposta.

4.1 ANÁLISE

4.1.1 Visão Geral

O agregador proposto foi desenvolvido de visando facilitar o acompanhamento de *feeds* (com enfoque em *feeds* de notícias) de forma a personalizar a ordem de apresentação das notícias para um usuário de acordo com o monitoramento da utilização passada da aplicação por parte deste usuário.

Utilizando uma técnica aplicada anteriormente a filtros bayesianos para a eliminação de SPAMs (correio eletrônico não-solicitado e comumente indesejável) em caixas postais de correio eletrônico, o software agregador procura antecipar o interesse do usuário em certas notícias através da análise do conteúdo da notícia, atribuindo à ela um valor, correspondente ao possível grau de interesse do usuário, utilizado então para classificar a ordem em que as notícias são apresentadas.

4.1.2 Solução Proposta

4.1.2.1 Aplicação do teorema de Bayes

O exemplo a seguir propõe-se a explicar a forma na qual o Teorema de Bayes é utilizado para a classificação de notícias no presente trabalho. Utilizando a combinação de novas evidências acerca de um evento, calcula-se a probabilidade de que este evento tenha ocorrido. Neste caso, deseja-se calcular a probabilidade de uma

notícia ser considerada interessante pelo usuário baseado no conteúdo presente na notícia.

Como forma de simplificação do problema e facilitar o cálculo das probabilidades, assume-se a independência da ocorrência entre as palavras encontradas em cada notícia. Tal abordagem é conhecida na literatura como *Classificador Bayesiano Ingênuo* (ou *Naive Bayes Classifier)* [DOMINGOS] [RISH] e apesar de independência entre as palavras não ser comumente verdadeira, este classificador mostra-se surpreendentemente efetivo, quando comparado a outros métodos de aprendizado de máquina mais complexos [VINOKOUROV] [ZHANG].

Para a aplicação desta abordagem no problema de classificação de notícias, fez-se uma primeira distinção entre o grau de interesse (ou desinteresse) do usuário em relação às notícias a serem analisadas, estabelecendo duas categorias possíveis: notícias *interessantes* e *não-interessantes*.

O ato de leitura, por parte do usuário, de uma determinada notícia, classifica-a como interessante para aquele usuário. Notícias não-lidas são consideradas como não-interessantes. Desta forma, para o universo de todas as notícias passíveis de serem apresentadas ao usuário, os eventos de uma noticia ser considerada interessante ou não-interessante são mutuamente exclusivos, formando uma partição deste universo. Assim, considerando novas evidências para uma determinada notícia (as palavras-chave que representam seu conteúdo), pode-se atualizar a probabilidade da notícia ser considerada *interessante* dada a observação dos eventos de ocorrência das palavras-chave presentes na notícia.

Considerando *X*, *Y* e *Z* o universo de três palavras possíveis de ocorrer em uma determinada noticia, pode-se simplificar, para fins de compreensão, o modelo proposto para classificação da seguinte forma (considerações para um usuário qualquer):

Número de notícias consideradas **interessantes**: 40

Número de notícias consideradas **não-interessantes**: 60

Número total de notícias avaliadas: 100

Notícias / Palavras	**X**	**Y**	**Z**
Ocorrência em notícias **interessantes**	20	8	16
Ocorrência em notícias **não-interessantes**	6	54	18

Notícias / Palavras	**X**	**Y**	**Z**
Frequência em notícias **interessantes**	50%	20%	40%
Frequência em notícias **não-interessantes**	10%	90%	30%

Através destas informações, é possível aplicar a fórmula do Teorema de Bayes para calcular o provável grau de interesse deste usuário quando surgirem novas notícias contendo uma ou mais destas palavras.

Considerando *I* o evento de uma notícia ser interessante e *I'* seu evento complementar, para a *palavra X* tem-se que:

$$P(I \mid X) = \frac{P(X \mid I) \cdot P(I)}{P(X \mid I) \cdot P(I) + P(X \mid I') \cdot P(I')} = 0{,}7692$$

Calculando o restante das probabilidades de interesse para as demais combinações possíveis obtém-se:

Provável grau de interesse		
P (interesse | X)	0,7692	76,92%
P (interesse | Y)	0,1290	12,90%
P (interesse | Z)	0,4706	47,06%
P (interesse | X, Y)	0,4255	42,55%
P (interesse | X, Z)	0,8163	81,63%

P (interesse | Y, Z)	0,1649	16,49%
P (interesse | X, Y, Z)	0,4969	49,69%

A utilização do agregador atualiza constantemente as freqüências das palavras-chave em notícias interessantes e não-interessantes. Com isto é possível "treinar" a aplicação de acordo com a utilização passada de cada usuário, calculando o possível grau de interesse para cada notícia nova trazida.

4.1.2.2 Extração de palavras-chave

Nem todas as palavras presentes em uma notícia são essenciais para a determinação de seu significado. Como forma de aprimorar o processo de cálculo do grau de interesse da notícia, palavras que apresentam um baixo conteúdo informacional, denominadas de *stopwords*, são removidas do conteúdo indexado no banco de dados.

Este processo de remoção de *stopwords* implica o fato de que algumas palavras possuem um peso maior de significância do que outras, para o processo de avaliação do conteúdo do texto [SOUZA]. Para tanto, houve a necessidade de criação de uma lista de *stopwords* a serem desconsideradas quando presentes em uma notícia. Esta lista, denominada de *stoplist*, é composta na maioria das vezes por termos que não identificam um contexto específico, sendo inerentes à linguagem e ao idioma e não ao conteúdo da notícia.

Através do estudo da gramática da língua portuguesa, selecionaram-se classes gramaticais que caracteristicamente compõem uma *stoplist*.

Artigos

São palavras que antepõem ao substantivo para determiná-lo ou indeterminá-lo, caracterizar-lhe nas categorias de gênero ou número; reduzir substantivos próprios a comuns; substantivar qualquer outra classe de palavras [LUFT].

Pronomes

São palavras que denotam o ser, como o (nome) substantivo, mas sem lhe dar a significação natural, ou referem-se ao ser, como o (nome) adjetivo, mas sem lhe apontar qualidade ou propriedade. Sua função é simplesmente indicar ou determinar o ente do ponto de vista de quem fala ou em relação à frase [LUFT].

Numerais

Palavras que denotam quantidade, ordem, proporção [LUFT]. Como os pronomes, o numera pode aparecer sozinho na frase ou junto de um substantivo.

Advérbios

São palavras de natureza nominal ou pronominal que na frase acrescentam relacionam-se à significação de um verbo, de um adjetivo, de outro advérbio ou de toda uma frase [LUFT].

Preposições

Palavras invariáveis que relacionam dois termos. Nessa relação, um termo completa ou explica o sentido do outro [LUFT].

Conjunções

São vocábulos invariáveis que estabelecem coordenação ou subordinação entre dois termos de uma oração, entre duas orações, entre um termo e uma oração, e mais raramente, entre dois períodos.

Interjeições

Palavra ou grupo de palavras com que exprimem emoções súbitas, sentimentos, idéias, não logicamente estruturadas. É a parte menos racional da linguagem.

É relevante enfatizar que uma mesma palavra pode ser de uma ou de outra classe, conforme o contexto. Assim, a classe de muitas palavras só se torna nítida em um contexto determinado [LUFT]. Portanto, apesar das vantagens, deve-se ter atenção durante a remoção de *stopwords* pois o processo pode comprometer o significado da notícia caso sejam eliminados termos relevantes à representação do texto.

4.1.3 Requisitos

4.1.3.1 Requisitos funcionais

O requisito fundamental do agregador é permitir ao usuário ler conteúdo distribuído *online* sob a forma de *feeds*. Para que isto seja possível, a aplicação deve suportar os formatos e as versões de *feeds* mais comumente usadas atualmente na Internet.

A aplicação também deve ser capaz de monitorar a utilização do agregador de notícias por parte de cada usuário, monitorando e armazenando registros para as notícias lidas e não lidas de cada *feed* assinado.

Por fim, a aplicação deve ser capaz de buscar notícias atualizadas na Internet para cada *feed* cadastrado no sistema, extraindo palavras-chave de cada notícia nova e salvando seu conteúdo (e suas respectivas palavras-chave) em um banco de dados de notícias, para acesso dos usuários.

4.1.3.2 Requisitos não funcionais

O agregador deve possuir uma interface web intuitiva e amigável. Um novo usuário da aplicação deve ser capaz de assinar novos *feeds* e ter acesso às suas notícias não lidas de maneira rápida e organizada.

O monitoramento realizado pela aplicação em relação às notícias lidas e não-lidas para cada usuário deve ser feito de forma não intrusiva: o usuário não deve sentir sua privacidade comprometida, mesmo que o monitoramento realizado seja constante.

4.2 DESENVOLVIMENTO

4.2.1 Ferramentas utilizadas

A aplicação foi desenvolvida utilizando versões gratuitas de ferramentas disponibilizadas pela Microsoft em seu pacote de desenvolvimento Visual Studio. Este ambiente de desenvolvimento facilita a implementação de softwares para o sistema operacional Windows, bem como aplicativos web e persistência em banco de dados. Embora estas versões gratuitas (denominadas *Express*) possuam certas restrições [MSDN(1)] quando comparadas às versões completas e pagas das mesmas

ferramentas, elas foram satisfatórias e cumpriram os requisitos necessários para o desenvolvimento deste projeto.

Para desenvolver o software que busca as notícias na Internet e salva-as no banco de dados foi utilizado o *Microsoft Visual C# 2005 Express Edition* [MSDN(2)], o que facilitou a implementação em ambiente Windows. Para a interface *web* do agregador de notícias, foi utilizado o *Microsoft Visual Web Developer 2005 Express Edition* [MSDN(3)], cujo ambiente integrado de desenvolvimento reduziu bastante o esforço necessário para programar e testar os recursos necessários para a interface principal. Por fim, a persistência dos dados foi feita utilizado o *Microsoft SQL Server 2005 Express Edition* [MSDN(4)], cuja integração com as outras duas ferramentas mencionadas é feita de forma transparente.

Vale ressaltar que a opção por uma família de produtos de desenvolvimento madura, com uma IDE única tanto para o desenvolvimento de aplicativos Windows como aplicativos web, ajudou a manter a coesão e a reutilização de classes em diversos pontos da aplicação, aliviando boa parte da sobrecarga existente em outros ambientes para realizar tarefas simples, como o desenvolvimento de interfaces por exemplo. Isto permitiu uma atenção maior à implementação adequada do padrão de comunicação entre a as camadas lógica, de persistência e apresentação.

4.2.2 Módulos da aplicação

A aplicação de forma geral consiste em dois módulos principais, que executam de forma independente. O primeiro módulo, denominado *Painel de controle*, é um software executável, para o sistema operacional Windows, desenvolvido para alimentar uma base de dados com notícias dos *feeds* cadastrados no sistema. O segundo módulo consiste em uma interface web, denominada *Aurora*, que se conecta na base de dados alimentada pelo *Painel de controle* e busca notícias ainda não lidas de *feeds* assinados por um usuário.

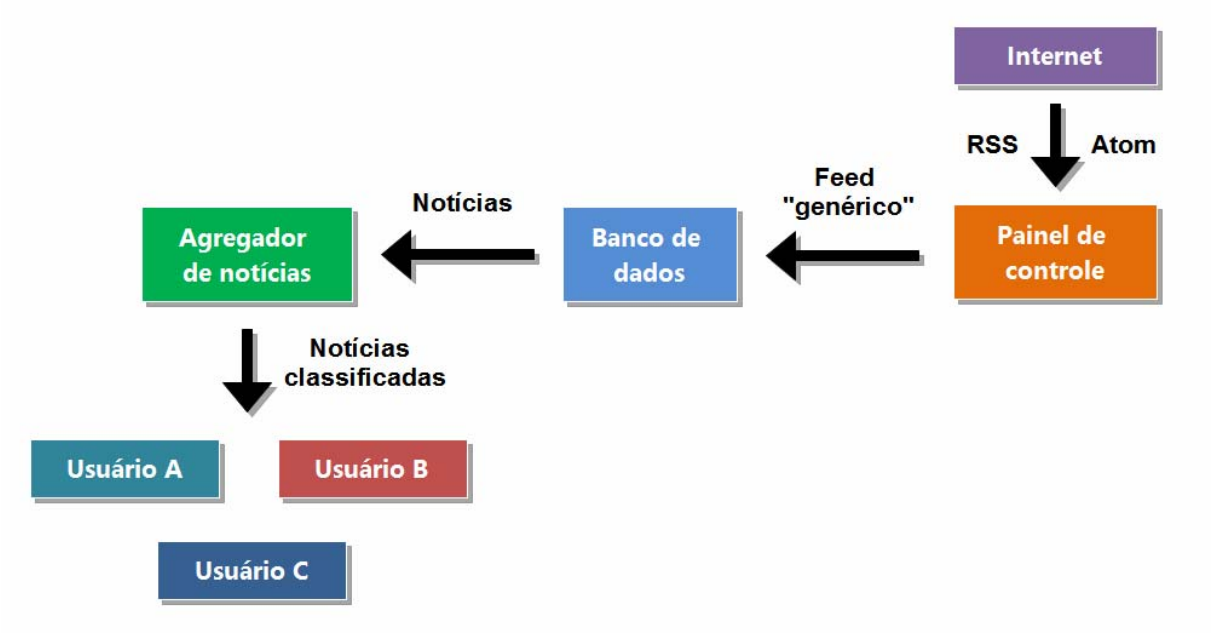

Figura 2. Visão geral da aplicação.

Em suporte a estes dois módulos encontra-se uma biblioteca de classes desenvolvida para realizar a análise, conversão e extração de palavras-chave de *feeds* e notícias, e um módulo *classificador*, que utiliza um classificador bayesiano para tentar antecipar o interesse do usuário em determinadas notícias.

O software *Painel de controle* é responsável por buscar novas notícias de forma automática, para *feeds* cadastrados no sistema. As notícias são então convertidas para um formato de *feed* genérico usado pela aplicação e salvas no banco de dados.

Em um segundo momento, um usuário acessa a interface *web* e obtém a lista de todas as notícias não-lidas para todos os *feeds* assinados por ele. Antes, porém, a interface web chama o módulo classificador para que este classifique a lista de notícias na provável ordem em que o usuário gostaria de lê-las. Somente então as notícias trazidas do banco de dados são apresentadas ao usuário.

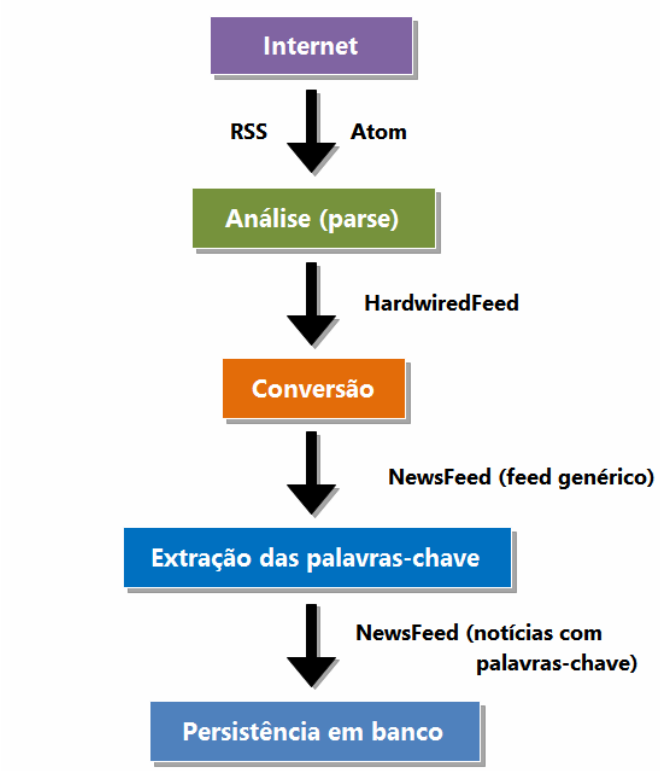

Figura 3. Atividades do *Painel de controle*.

4.2.3 Camada lógica

4.2.3.1 Biblioteca de classes para suporte a feeds

De acordo estatísticas fornecidas pelo *website* **Syndic8.com** [SYNDIC8], responsável por cadastrar e catalogar mais de 500.000 *feeds* na Internet, 80,55% dos *feeds* cadastrados estão no formato RSS. Destes, 81,45% correspondem à versão 2.0 do RSS; a versão 0.91 é utilizada em 9,41% dos *feeds* e as versões 1.0 e 0.92 possuem 7,16% e 1,36%, respectivamente. O restante (menos de 1%) dos *feeds* neste formato são versões de baixo uso historicamente, com pouca aderência por parte dos usuários.

O formato Atom, que possui uma parcela de 19,44% de uso total entre os dois formatos, possui atualmente apenas a versão 1.0 cujo uso é recomendado. Embora sua versão anterior (0.3) ainda possa ser encontrada em uso em alguns *feeds*, foi

depreciada em favor de sua versão mais nova e sua utilização e suporte por parte dos agregadores é desencorajada pelos mantenedores da especificação Atom [*FEED*].

O agregador de noticias *Aurora*, através de sua biblioteca de classes, desenvolvida para manipulação de *feeds*, suporta as versões 0.91, 0.92, 1.0 e 2.0 do formato RSS e a versão 1.0 do formato Atom. Com isto, ele abrange praticamente 100% dos formatos mais populares atualmente usados, permitindo o acesso dos usuários ao conteúdo atualizado disponível sob a forma de *feeds* na Internet.

Um dos problemas decorrentes da existência de várias versões do formato RSS em uso, bem como de dois diferentes formatos para *feeds* – RSS e Atom – é que isto se traduz no suporte à estas múltiplas versões dentro do agregador, considerando as peculiaridades de cada especificação. Como uma forma de minimizar o impacto do surgimento de novas versões para os formatos suportados e também do possível surgimento de novos formatos de *feeds*, desde o início do desenvolvimento da biblioteca de classes optou-se por criar e utilizar um modelo "genérico" de *feed*, que possuísse as características mais comuns dos formatos suportados.

Desta forma, através da análise das especificações de cada formato e das diferentes versões (para o caso do RSS) chegou-se à um denominador comum sobre os atributos mais relevantes em cada especificação que deveriam ser levados em consideração no momento da análise do conteúdo dos *feeds*. Isto culminou na criação de uma classe de *feed* genérica, para uso interno da aplicação, que é devolvida pela biblioteca depois de feita a análise e conversão do conteúdo específico de cada *feed*.

Com isto, garantiu-se a simplicidade no uso da biblioteca de classes, bastando ao desenvolvedor fornecer a URL do *feed* desejado e receber como retorno um objeto de *feed* genérico, no qual os diferentes formatos de *feeds* não têm mais importância, mas sim o conteúdo carregado pelo objeto. Desta forma também foi possível estabelecer um padrão para as informações carregadas entre as camadas da aplicação, facilitando tanto a apresentação do conteúdo para o usuário quanto a persistência dos dados no banco.

Análise

O primeiro passo realizado no processo de análise consiste em buscar o conteúdo do *feed* solicitado. Este pode estar localizado tanto na Internet quanto em um arquivo acessível através da máquina na qual o software *Painel de controle* está sendo executado.

Após o conteúdo do *feed* ter sido carregado, seu *stream* XML é passado para a classe *FeedParser* que realiza a análise do conteúdo do *feed* através de uma chamada do método *Parse*. O processo de análise (*parse*) de um *feed* consiste primeiramente em analisar o cabeçalho do XML passado e identificar o formato e a versão do *feed*. A classe *FeedParser* então realiza a chamada do método de análise da classe responsável por aquele tipo especifico de *feed*. O *Aurora* possui as classes de análise *Rss091Parser*, *Rss092Parser*, *Rss10Parser* e *Rss20Parser* para as versões 0.91, 0.92, 1.0 e 2.0 do formato RSS; para o formato Atom, existe a classe *Atom10Parser*, responsável pela versão 1.0 deste formato. Cada uma destas classes conhece a lógica por trás da especificação do formato tratado por ela. Portanto, são elas as responsáveis por *extrair* as informações do *stream* XML trazido da Internet.

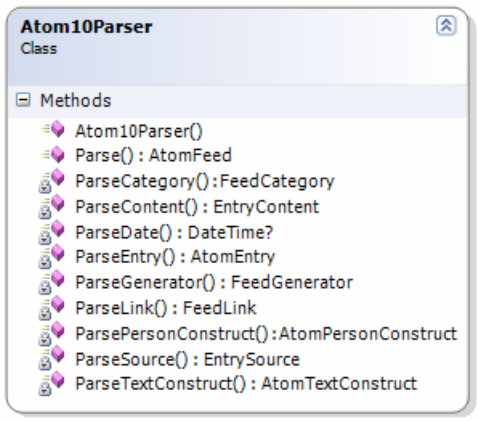

Figura 4. Classe de análise para o formato Atom.

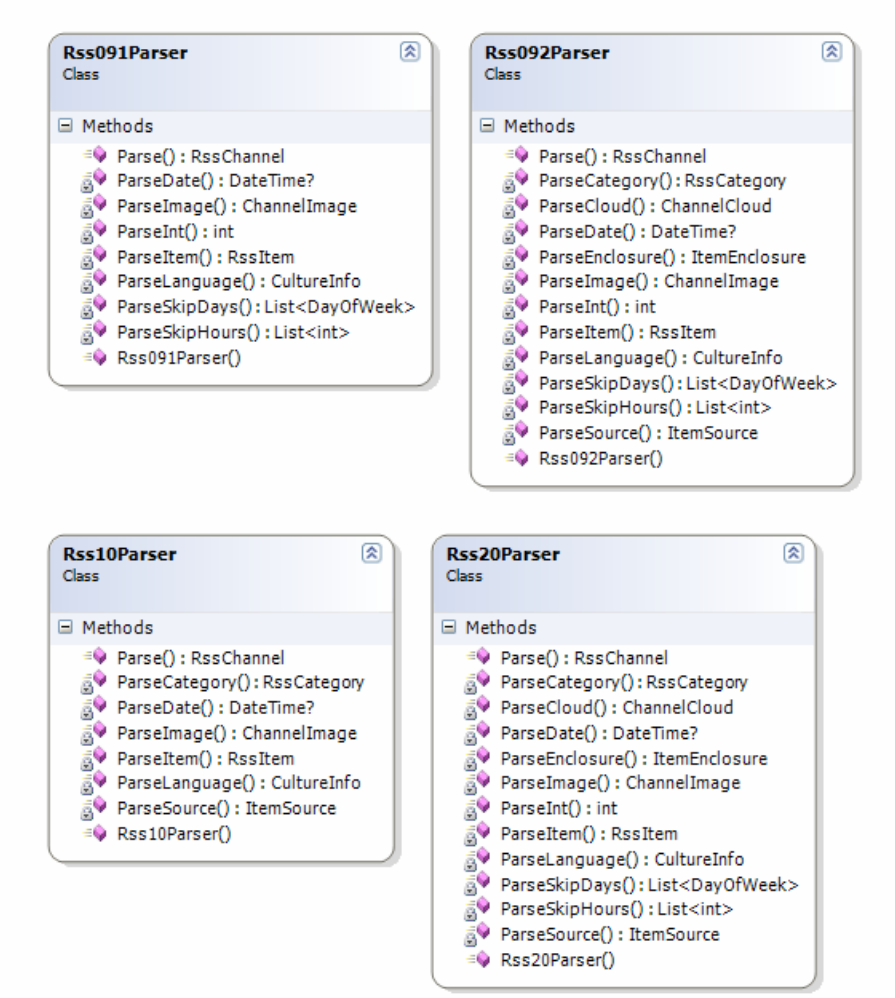

Figura 5. Classes de análise para o formato RSS.

As informações extraídas por cada classe de análise são inseridas em objetos que representam *feeds* de formatos específicos. Após a análise de um *feed* RSS, não importando a versão, a classe responsável pela análise feita devolve um objeto do tipo *RssChannel*; de forma similar, após a análise de um *feed* Atom um objeto do tipo *AtomFeed* é retornado.

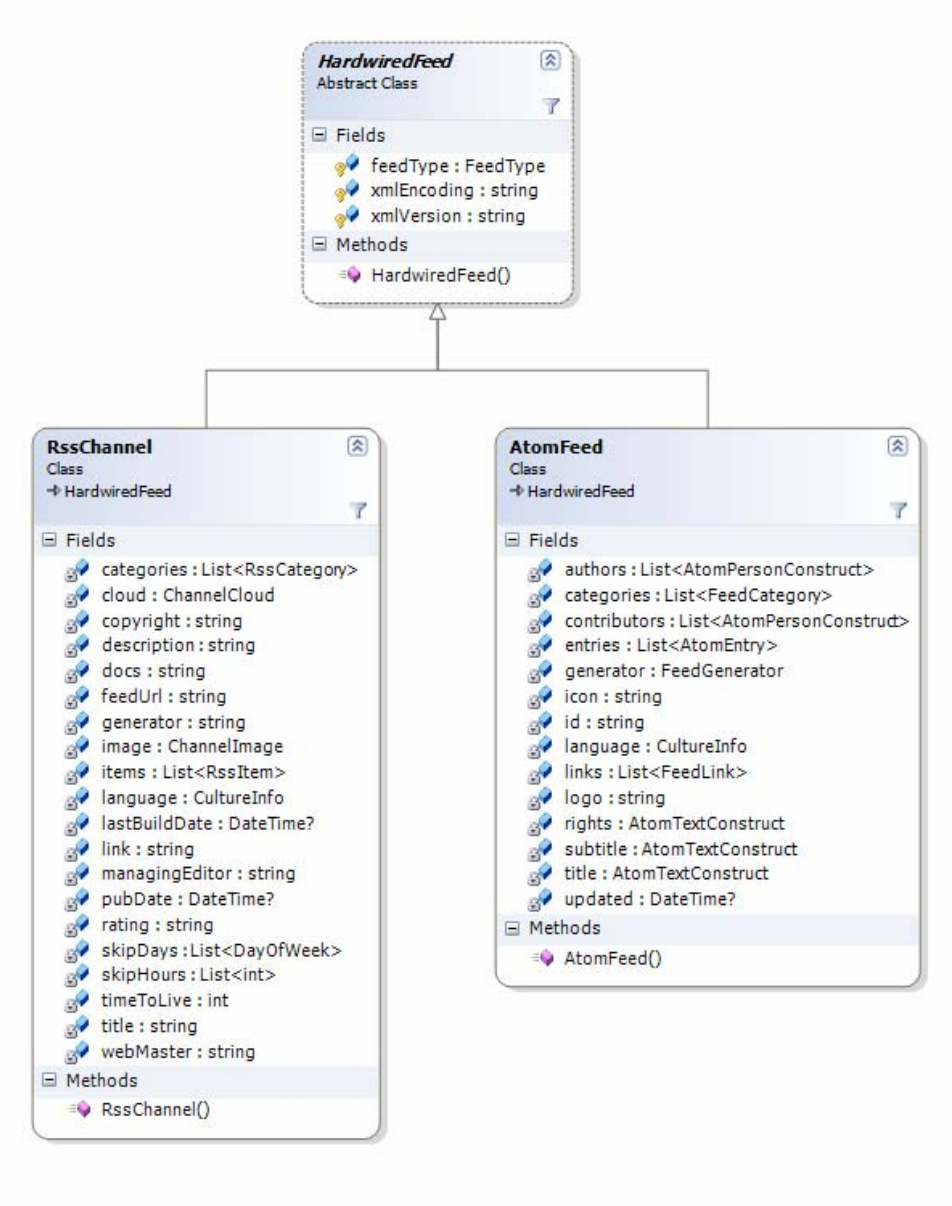

Figura 6. Classes de mapeamento dos formatos específicos de *feeds*.

Ambas as classes derivam da classe abstrata *HardwiredFeed*, que representa um *feed* cujo conteúdo ainda está atrelado à um formato específico. Estes *feeds* de formato específico são então devolvidos pela classe *FeedParser* para a classe principal da biblioteca de tratamento de *feeds*, *FeedReader*.

Conversão

Como mencionado anteriormente, os objetos *HardwiredFeed* devolvidos pela classe *FeedParser* ainda são atrelados à formatos específicos e não correspondem à idéia de um formato genérico de *feed*, necessário para trafegar informação entre as camadas da aplicação. Com isto torna-se necessário *converter* o objeto *feed* de formato específico para um objeto genérico. Isto é feito através da classe *FeedConverter*.

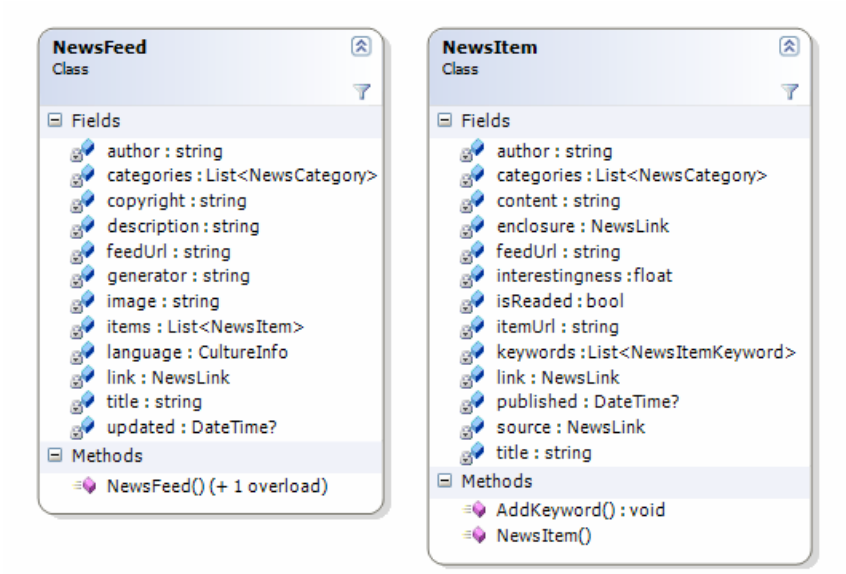

Figura 7. Classes do formato genérico de *feed*.

Internamente a classe *FeedConverter* possui duas classes responsáveis por cada um dos formatos a serem convertidos: as classes *RssConverter* e *AtomConverter*. Cada uma destas é responsável por receber objetos *HardwiredFeed* do tipo *RssChannel* e *AtomFeed*, respectivamente, e convertê-los para o formato de *feed* genérico *NewsFeed*. A classe *AtomConverter* converte o objeto de *feed* específico do tipo Atom diretamente para um objeto genérico *NewsFeed*. A classe *RssConverter*, entretanto, aproveita-se da semelhança entre as diversas versões do formato RSS e

converte em um único passo os diversos formatos RSS existentes para objetos *NewsFeed*.

Extração de palavras-chave

O processo de obtenção de um *feed* genérico para o uso da aplicação é feito através dos procedimentos de análise e conversão, já descritos. Porém, um terceiro passo é necessário para tornar o objeto de *feed* genérico útil para sua utilização no processo de *classificação* das notícias que ocorre momentos antes de serem exibidas ao usuário. Este passo consiste na extração das palavras-chave consideradas relevantes em cada notícia.

Cada *feed* no formato genérico *NewsFeed* possui como atributo uma lista de notícias, também em formato genérico, da classe *NewsItem*. Objetos de notícia *NewsItem* por sua vez, possuem como atributo uma lista de palavras-chave, da classe *NewsItemKeyword*.

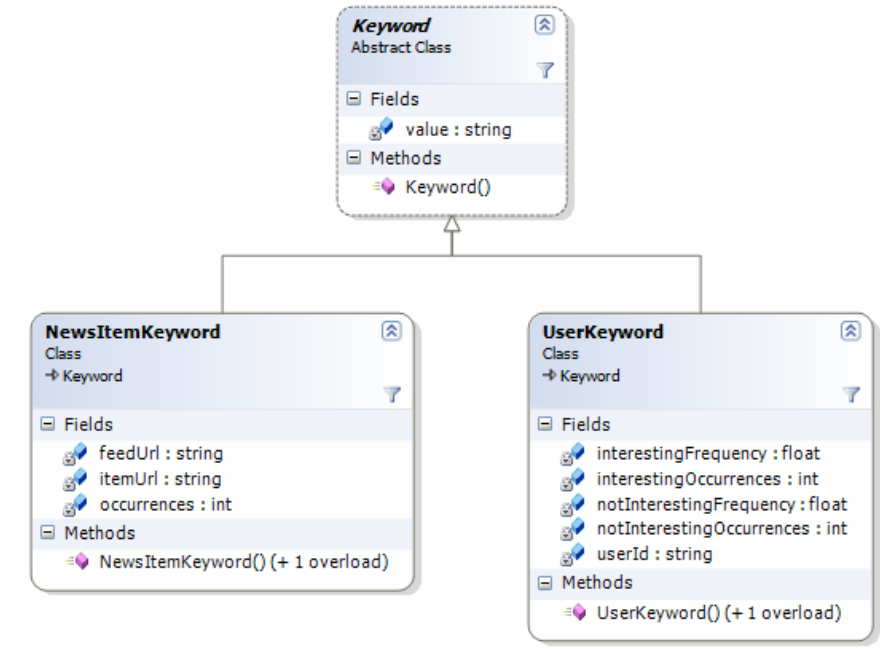

Figura 8. Classes para as palavras-chave da aplicação.

O processo de extração de palavras-chave, de forma geral, consiste primeiramente em normalizar o conteúdo de uma notícia, remover as palavras consideradas indesejáveis e então adicionar, uma a uma, as palavras restantes como objetos *NewsItemKeyword* para uma determinada notícia. O objeto retornado pela classe *KeywordExtractor* é basicamente o mesmo objeto *NewsFeed* recebido como parâmetro, com a diferença de cada notícia retornada possui uma lista de palavras-chave dentro dela que a representa.

Normalização

Formatos de *feeds* como Atom e RSS 2.0 suportam a utilização de marcações HTML para enriquecer a semântica e a apresentação do conteúdo distribuído *online*. Além disso, sinais de pontuação, algarismos e caracteres diversos que não correspondem à letras permeiam os textos de conteúdo das notícias. Assim, antes de executar o procedimento de isolamento das palavras relevantes em uma notícia, mostrou-se necessário efetuar uma "normalização" do conteúdo, removendo qualquer elemento que pudesse a prejudicar o processo de análise das palavras da notícia.

Esta normalização consiste na execução em seqüência, para cada notícia, de uma série de expressões regulares escritas para remover trechos não necessários à análise das palavras. Em um primeiro momento todas as marcações HTML são removidas, restando apenas o conteúdo em forma de texto. Em seguida, todos os caracteres que não sejam letras, números e hífen são removidos. A seguir, números, caracteres isolados e espaços em branco são também removidos, restando apenas palavras com duas ou mais letras separadas por um único espaço em branco entre si. Por fim, todas as palavras são convertidas para letras minúsculas e devolvidas como uma lista de *strings* para o passo seguinte do processo de extração.

Remoção via stoplist

Como mencionado anteriormente, uma maneira de extrair as palavras-chave de um determinado texto é remover de seu conteúdo as palavras indesejáveis. Através da utilização de uma *stoplist*, o algoritmo de extração percorre toda a lista de palavras restantes para o conteúdo da notícia e remove aquelas que pertencem à *stoplist*. Ao final do processo, palavras comuns da língua portuguesa, pertencentes a classes gramaticais que não acrescentam significado semântico à notícia são removidas. As palavras restantes são consideradas pela aplicação como **palavras-chave** que representam de forma geral o principal conteúdo da notícia. Estas palavras-chave são essenciais para a classificação futura de cada notícia, sendo portanto indexadas no banco de dados juntamente com cada notícia trazida para o *feed*.

4.2.3.2 Classificador bayesiano

O processo de classificação de uma notícia, como já explicado anteriormente, leva em consideração evidências (palavras-chave) presentes em cada notícia a ser classificada. Com os valores de frequência de cada palavra-chave em notícias interessantes e não-interessantes sendo atualizados a todo instante a partir da utilização da aplicação por parte do usuário, o processo de classificação das notícias deve ocorrer momentos antes das mesmas serem disponibilizadas para a leitura do usuário.

A classe responsável pela classificação de uma lista de notícias, denominada de *BayesianClassifier*, recebe como argumento ou um objeto *NewsFeed* ou uma lista de objetos *NewsItem* a serem classificados. Para o caso de ter sido passado um objeto *NewsFeed*, o classificador utiliza a lista de *NewsItem* dentro dele. Cada objeto "genérico" de notícia *NewsItem* a ser classificado possui como atributo uma lista de palavras-chave que representam a notícia.

O primeiro passo do classificador é obter as freqüências de ocorrência de cada palavra-chave, para cada notícia, em notícias interessantes e não-interessantes. Portanto, para cada notícia, o classificador solicita do banco de dados estes valores de frequência em relação ao usuário atual, recebendo-os como uma lista de objetos *UserKeyword*. A seguir, o classificador solicita a frequência total de notícias interessantes e não-interessantes avaliadas pelo usuário atual. De posse destes valores, o classificador pode então, através da aplicação do Teorema de Bayes, calcular o possível grau de interesse do usuário baseado nas palavras-chave presentes na notícia. Palavras-chave presentes mais de uma vez são consideradas de acordo com sua frequência na notícia, elevando ou diminuindo a probabilidade da notícia ser considerada interessante.

Após ter sido obtido o valor da probabilidade, o classificador insere este valor *dentro* da notícia sendo avaliada e o processo se repete para todas as outras notícias presentes no *feed*. Ao final do processo, o classificador devolve o mesmo *feed* passado como argumento, porém com os valores do possível grau de interesse de cada notícia presente em todas elas. É interessante notar que o classificador não ordena as notícias; ele apenas atribui a elas um valor específico que pode então ser usado posteriormente para alterar a maneira de apresentar as notícias de acordo.

4.2.4 Camada de apresentação

4.2.4.1 Painel de controle

O software *Painel de controle* desenvolvido tem o propósito de alimentar uma base de dados com notícias trazidas da Internet. O usuário final da aplicação não interage diretamente com este módulo da aplicação, sendo seu uso reservado para o administrador responsável pelo agregador de notícias.

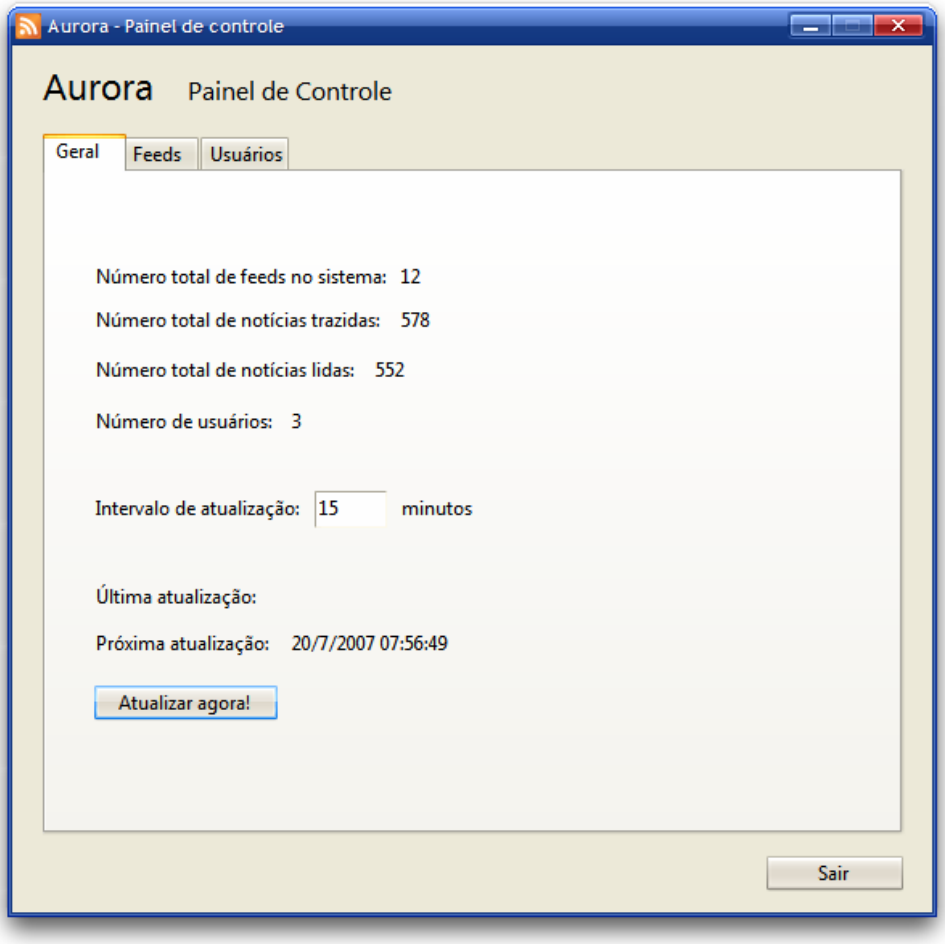

Figura 9. Software *Painel de controle*.

O *Painel de controle* é uma aplicação Windows que executa em *background* em uma máquina servidor, conectando-se à Internet a cada intervalo de tempo pré-determinado, em busca de novas notícias para todos os *feeds* assinados por usuários do sistema. Para cada *feed* cadastrado, o *Painel de controle* realiza uma chamada da classe *FeedReader* passando como parâmetro a URL do *feed* que necessita ser atualizado. A biblioteca de classes para o tratamento de *feeds* então se responsabiliza por conectar-se à Internet e manipular o *stream* XML do *feed*, devolvendo ao *Painel de controle* um objeto *NewsFeed* com o conteúdo necessário.

Para cada objeto *NewsFeed* recebido, o *Painel de controle* atualiza o banco de dados com as novas notícias presentes em cada *feed* analisado. Desta forma, sempre

que um usuário solicita a leitura de novas notícias, garante-se um intervalo máximo de defasagem entre a distribuição do conteúdo *online* e sua inclusão no banco de dados de notícias.

4.2.4.2 Agregador de notícias

A implementação principal do trabalho consiste no agregador de notícias *Aurora*, responsável por exibir as notícias não lidas para um determinado usuário. Optou-se pelo desenvolvimento de uma interface simples para o agregador em um primeiro momento, com todas as notícias não lidas de todos os *feeds* assinados por um usuário sendo exibidas como uma lista única.

O agregador consiste basicamente em uma página *web*, que exibe uma listagem de dez notícias de cada vez. É dada ao usuário a possibilidade de recarregar a lista com novas notícias não lidas, até que todas as notícias disponíveis para leitura em um determinado momento sejam mostradas.

O agregador é o responsável pela ordenação das notícias de acordo como foram classificadas. A página que exibe as notícias para o usuário solicita ao banco de dados uma lista de objetos *NewsFeed* contendo todas as notícias não lidas para todos os *feeds* assinados pelo usuário. A seguir, todas as notícias não lidas são reunidas em uma única lista (de objetos *NewsItem*) que é então passada para a classe *BayesianClassifier*, que realizará a classificação.

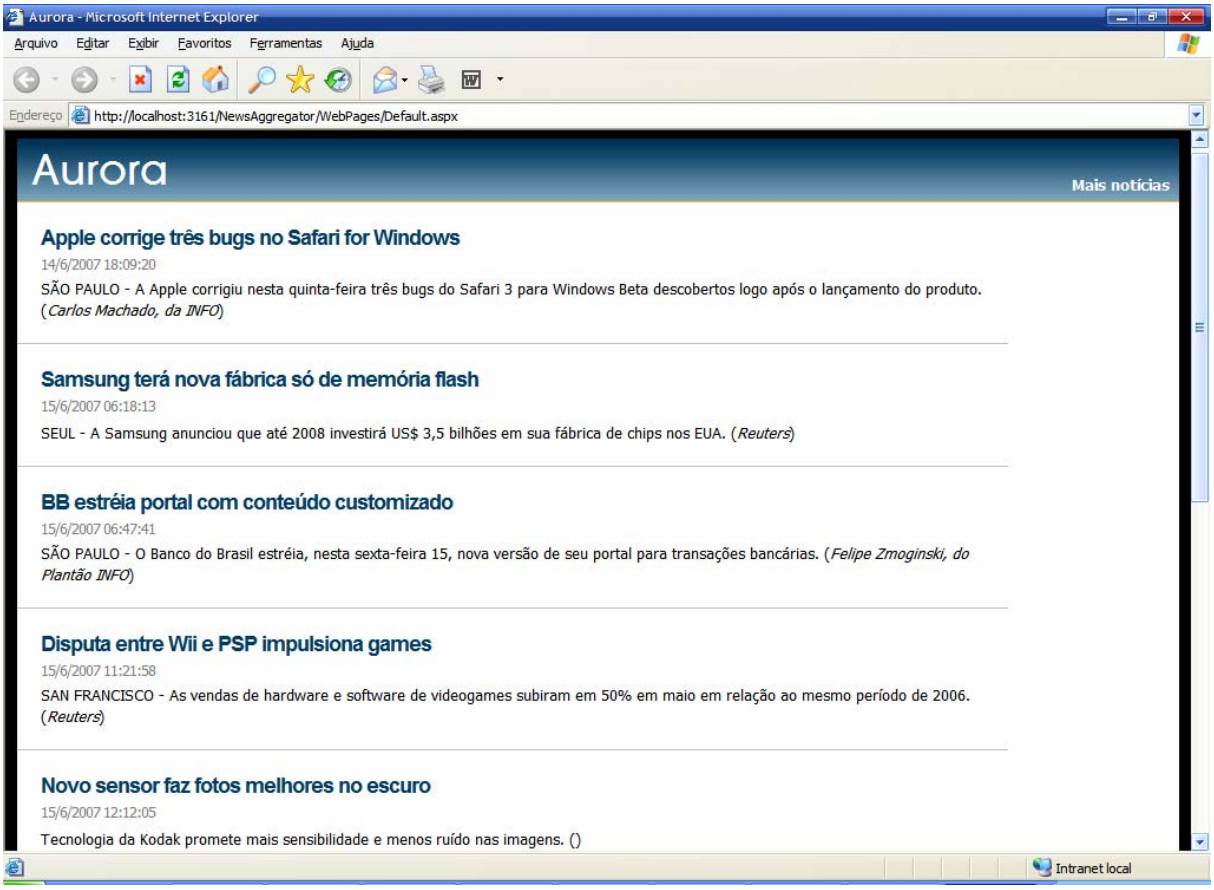

Figura 10. Agregador de notícias *Aurora*.

Assim que recebe de volta a lista de notícias com suas respectivas probabilidades, a página *web* do agregador ordena a lista de acordo com os valores calculados para cada notícia. Esta é considerada pela aplicação a ordem ideal para apresentação das notícias ao usuário.

Por fim, de posse da lista de notícias ordenadas, o agregador exibe-as para o usuário e então aguarda, esperando receber o *feedback* da leitura (ou não) das notícias mostradas.

O ato de leitura de uma notícia dispara um evento de *postback* da página web que exibe as notícias. Neste momento, no servidor, todas as palavras-chave pertencentes à notícia lida têm sua frequência em notícias *interessantes* atualizadas e em seguida uma nova janela do navegador é aberta, exibindo o conteúdo da notícia para o usuário.

De forma análoga, cada vez que o usuário solicita a exibição de mais notícias, todas as palavras-chave de todas as notícias não-lidas que estavam sendo mostradas têm seus valores de frequência em notícias *não-interessantes* atualizados e então uma nova leva de dez notícias não lidas é mostrada.

Este processo repete-se durante a utilização do agregador pelo usuário e os valores de frequência em notícias interessantes e não interessantes vão sendo atualizados, para uma posterior classificação feita para novas notícias que sejam adicionadas ao banco de dados.

5 CONCLUSÃO

A utilização do teorema de Bayes se mostrou uma solução viável quando aplicado ao problema de classificação de notícias. O baixo custo de implementação do algoritmo aliado aos poucos parâmetros necessários para obter um valor com o qual se pode inferir um possível interesse do usuário em uma determinada notícia mostraram-se ideais para utilização computacional do teorema em uma aplicação prática.

Entretanto, um aspecto importante que se destacou posteriormente no trabalho foi a importância na escolha das palavras-chave a serem utilizadas no processo de classificação. Optou-se por um método simples de eliminação de palavras-chave desnecessárias, através da utilização de *stopwords*. Porém, analisando diretamente as palavras-chave inseridas no banco de dados, percebe-se a existência de "ruído" – palavras-chave presentes em cada notícia, que podem levar a um cálculo errôneo do grau de interesse na notícia. Como exemplos de tais palavras-chave, podem ser citados nomes de agências de notícias ("Reuters") ou o nome de cidades ("São Paulo"), que aparecem com grande frequência em diferentes notícias, mas não se relacionam diretamente ao seu conteúdo. Tais palavras-chave podem representar um desvio no valor calculado para a probabilidade de interesse, apesar de testes posteriores serem necessários para comprovar tal efeito.

É interessante também observar que o desenvolvimento da biblioteca de classes para suporte a *feeds* para o *framework* .NET contribui para a comunidade de desenvolvedores desta plataforma. Até o presente trabalho, existiam apenas bibliotecas isoladas para tratamento de formatos específicos (RSS ou Atom) que não dispunham da flexibilidade de um formato de *feed* "genérico" para ser utilizado internamente na aplicação. Com a presente implementação, o problema do tratamento das diferentes versões e diferentes formatos é drasticamente reduzido, possibilitando ao desenvolvedor focar-se em na implementação de aplicativos que melhor utilizem o *conteúdo* extraído dos formatos de *feeds* mais utilizados atualmente na Internet.

5.1 TRABALHOS FUTUROS

Como já citado anteriormente, um ponto interessante passível de ser explorado é a definição detalhada das palavras-chave a serem extraídas do conteúdo das notícias, de forma que se elimine (ou pelo menos minimize) o número de palavras-chave desinteressantes para o processo de classificação. A análise de *corpus* de textos de língua portuguesa, de seus aspectos sintáticos, semânticos e características de frequência, pode resultar em melhorias, não só para o classificador bayesiano, mas também para outros algoritmos que baseiam seus cálculos na presença de palavras-chave no conteúdo a ser classificado.

Outra possibilidade seria a criação de perfis para os usuários, baseados nas palavras-chave já analisadas para cada um. Desta forma seria possível aplicar análises nos perfis de diferentes usuários e traçar comparativos, sendo possível eventualmente sugerir notícias ou mesmo *feeds* ainda não lidos para um determinado usuário, de acordo com a semelhança existente entre seus perfil e o de outros usuários.

6 REFERÊNCIAS BIBLIOGRÁFICAS

ANDREESSEN, M. **Innovators of the net: Ramanathan V. Guha and RDF.** Netscape Communications Corporation. Disponível em: <http://wp.netscape.com/columns/techvision/innovators_rg.html> Acesso em: 03, jun. 2007.

BARBETTA, P. A. **Estatística aplicada às Ciências Sociais.** 4. ed. Florianópolis: Ed. da UFSC, 2001.

BASTOS, L. R. M. **Apostila de Probabilidade e Estatística.** Rio de Janeiro: Universidade Estácio de Sá, 2005.

BRAY, T.; PAOLI, J.; SPERBERG, C. M.; MALER, E.; YERGEAU, F. **Extensible Markup Language (XML) 1.0 (Fourth Edition).** World Wide Web Consortium. Disponível em: <http://www.w3.org/TR/REC-xml/> Acesso em: 03, jun. 2007.

DOMINGOS, P.; PAZZANI, M. J. **On the Optimality of the Simple Bayesian Classifier under Zero-One Loss**, Kluwer Academic Publishers - Hingham, MA, USA., v. 29, p. 103-130, 1997.

FEED VALIDATOR NEWS, **Atom 0.3 Support Deprecated.** *Feed* Validator. Disponível em: <http://www.atomenabled.org/*feed*validator/news/archives/2005/09/15/atom_03_deprecated.html> Acesso em: 03, jun. 2007.

GUHA, R. V.; BRAY, T. **Meta Content Framework Using XML.** World Wide Web Consortium. Disponível em: <http://www.w3.org/TR/NOTE-MCF-XML/> Acesso em: 03, jun. 2007.

HAMMERSLEY, B. **Content Syndication with RSS.** O'Reilly & Associates, 2003.

IETF. **Atom Publishing Format and Protocol (atompub).** IEFT – The Internet Engineering Task Force. Disponível em: <http://www.ietf.org/html.charters/atompub-charter.html/> Acesso em: 03, jun. 2007.

IETF(2). **The Atom Syndication Format.** IEFT – The Internet Engineering Task Force. Disponível em: <http://www.ietf.org/rfc/rfc4287.txt> Acesso em: 03, jun. 2007.

KING, A. **The Evolution of RSS.** WebReference.com. Disponível em: <http://www.webreference.com/authoring/languages/xml/rss/1/> Acesso em: 03, jun. 2007.

LIBBY, D. **RSS 0.91 Spec, revision 3.** Internet Archive: Wayback Machine. Disponível em: <http://web.archive.org/web/20001204093600/my.netscape.com/publish/formats/rss-spec-0.91.html> Acesso em: 03, jun. 2007.

LUFT, C. P.; **Novo manual de português, gramática, ortografia oficial, literatura, redação, textos e testes.** Nova edição revista e atualizada 1995.

MEYER, P. L. **Probabilidade:** aplicações à estatística. 2. ed. Rio de Janeiro: LTC – Livros Técnicos e Científicos Editora S.A., 1983.

MNN - MY NETSCAPE NETWORK. **My Netscape Network Help.** Internet Archive: Wayback Machine. Disponível em: <http://web.archive.org/web/20001208063100/http://my.netscape.com/publish/help/quickstart.html> Acesso em: 03, jun. 2007.

MSDN(1), **Visual Studio 2005 Product Line Overview.** MSDN Home. Disponível em: <http://msdn2.microsoft.com/en-us/vstudio/aa700921.aspx> Acesso em: 03, jun. 2007.

MSDN(2), **Visual C# Express – Product Overview.** MSDN Home. Disponível em: <http://msdn.microsoft.com/vstudio/express/visualcsharp/default.aspx> Acesso em: 03, jun. 2007.

MSDN(3), **Visual Web Developer – Product Overview.** MSDN Home. Disponível em: <http://msdn.microsoft.com/vstudio/express/vwd/default.aspx> Acesso em: 03, jun. 2007.

MSDN(4), **SQL Server Express – Product Overview.** MSDN Home. Disponível em: <http://msdn.microsoft.com/vstudio/express/sql/default.aspx> Acesso em: 03, jun. 2007.

RDF CORE WORKING GROUP. **Resource Description Framework (RDF).** World Wide Web Consortium. Disponível em: < http://www.w3.org/RDF/> Acesso em: 03, jun. 2007.

RISH, I. **An empirical study of the naive Bayes classifier**, T.J. Watson Research Center - Hawthorne, NY, USA.

RSS-DEV WORKING GROUP. **RDF Site Summary (RSS) 1.0.** web.resource.org. Disponível em: <http://web.resource.org/rss/1.0/spec> Acesso em: 03, jun. 2007.

RUBY, S., D. **Project Roadmap.** The Atom Project. Disponível em: <http://intertwingly.net/moin-1.2.1/wiki/cgi-bin/moin.cgi/RoadMap/> Acesso em: 03, jun. 2007.

SILVA, P. A. L. **Probabilidades & Estatística.** Rio de Janeiro: Reichmann & Affonso Editores, 1999.

SOUZA, R. R. **Uma proposta de metodologia para escolha automática de descritores utilizando sintagmas nominais.** Belo Horizonte, 2005. 215f. Tese apresentada ao Programa de Pós-Graduação em Ciência da Informação da Universidade Federal de Minas Gerais como requisito parcial à obtenção do título de Doutro em Ciência da Informação – Escola de Ciência da Informação, Universidade Federal de Minas Gerais.

SYNDIC8, **Syndic8.com – Site Statistics.** Syndic8.com. Disponível em: <http://www.syndic8.com/stats.php> Acesso em: 03, jun. 2007.

VINOKOUROV, A. **The Organisation and Retrieval of Document Collections:** A Machine Learning Approach. Paisley, Scotland, 2003. 213 f. Dissertation submitted in partial fullfilment of the requirements for the Degree of Doctor of Philosophy – School of Information and Communication Technologies, University of Paisley.

WINER, D. **Scripting News in XML.** DaveNet. Disponível em: <http://www.scripting.com/davenet/1997/12/15/scriptingNewsInXML.html> Acesso em: 03, jun. 2007.

WINER(2), D. **RSS 0.91 Copyright and disclaimer.** Backend.UserLand.Com. Disponível em: <http://backend.userland.com/rss091#copyrightAndDisclaimer> Acesso em: 03, jun. 2007.

WINER(3), D. **RSS 0.92.** Backend.UserLand.Com. Disponível em: <http://backend.userland.com/RSS092/> Acesso em: 03, jun. 2007.

YUDKOWSKY, E. **An Intuitive Explanation of Bayesian Reasoning.** Disponível em: <http://yudkowsky.net/bayes/bayes.html/> Acesso em: 03, jun. 2007.

ZHANG, H.; SU, J. **Naive Bayesian Classifiers for Ranking**. In: EUROPEAN CONFERENCE ON MACHINE LEARNING (ECML2004), 15., 2004, Pisa, Italy. **Proceedings of…**, Springer, 2004.

APÊNDICE 1 – CÓDIGO-FONTE

HardwiredFeed.cs

```
using System;
using System.Data;
using System.Configuration;

namespace Aurora.Feed.Types
{
    public abstract class HardwiredFeed
    {
        #region Attributes

        protected string xmlVersion;
        protected string xmlEncoding;
        protected FeedType feedType;

        #endregion

        #region Properties

        public string XmlVersion
        {
            get { return this.xmlVersion; }
            set { this.xmlVersion = value; }
        }

        public string XmlEncoding
        {
            get { return this.xmlEncoding; }
            set { this.xmlEncoding = value; }
        }

        public FeedType FeedType
        {
            get { return this.feedType; }
            set { this.feedType = value; }
        }

        #endregion

        #region Constructor

        public HardwiredFeed()
        {
            //construtor da classe HardwiredFeed;
        }

        #endregion
    }
}
```

RssChannel.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Collections.Generic;
using System.Globalization;
```

```
namespace Aurora.Feed.Types.Rss
{
    /// <summary>
    /// Classe que armazena os dados extraídos através do parser para o feed RSS.
    /// Suporta as versões 0.91, 0.92, 1.0 e 2.0 do formato RSS.
    /// </summary>
    public class RssChannel : HardwiredFeed
    {
        #region Attributes

        private string feedUrl;
        private string title;
        private string link;
        private string description;
        private CultureInfo language;
        private ChannelImage image;
        private string copyright;
        private string generator;
        private Nullable<DateTime> lastBuildDate;
        private Nullable<DateTime> pubDate;
        private string managingEditor;
        private List<RssCategory> categories;
        private List<RssItem> items;
        private string webMaster;
        private string rating;
        private int timeToLive;
        private ChannelCloud cloud;
        private string docs;
        private List<int> skipHours;
        private List<System.DayOfWeek> skipDays;

        #endregion

        #region Properties

        public string FeedUrl
        {
            get { return this.feedUrl; }
            set { this.feedUrl = value; }
        }

        public string Title
        {
            get { return this.title; }
            set { this.title = value; }
        }

        public string Link
        {
            get { return this.link; }
            set { this.link = value; }
        }

        public string Description
        {
            get { return this.description; }
            set { this.description = value; }
        }

        public CultureInfo Language
        {
            get { return this.language; }
```

```
    set { this.language = value; }
}

public ChannelImage Image
{
    get { return this.image; }
    set { this.image = value; }
}

public string Copyright
{
    get { return this.copyright; }
    set { this.copyright = value; }
}

public string Generator
{
    get { return this.generator; }
    set { this.generator = value; }
}

public Nullable<DateTime> LastBuildDate
{
    get { return this.lastBuildDate; }
    set { this.lastBuildDate = value; }
}

public Nullable<DateTime> PubDate
{
    get { return this.pubDate; }
    set { this.pubDate = value; }
}

public string ManagingEditor
{
    get { return this.managingEditor; }
    set { this.managingEditor = value; }
}

public List<RssCategory> Categories
{
    get { return this.categories; }
    set { this.categories = value; }
}

public List<RssItem> Items
{
    get { return this.items; }
    set { this.items = value; }
}

public string WebMaster
{
    get { return this.webMaster; }
    set { this.webMaster = value; }
}

public string Rating
{
    get { return this.rating; }
    set { this.rating = value; }
}
```

```
        public int TimeToLive
        {
            get { return this.timeToLive; }
            set { this.timeToLive = value; }
        }

        public ChannelCloud Cloud
        {
            get { return this.cloud; }
            set { this.cloud = value; }
        }

        public string Docs
        {
            get { return this.docs; }
            set { this.docs = value; }
        }

        public List<int> SkipHours
        {
            get { return this.skipHours; }
            set { this.skipHours = value; }
        }

        public List<System.DayOfWeek> SkipDays
        {
            get { return this.skipDays; }
            set { this.skipDays = value; }
        }

        #endregion

        #region Constructor

        public RssChannel()
        {
            categories = new List<RssCategory>();
            items = new List<RssItem>();
            skipHours = new List<int>();
            skipDays = new List<DayOfWeek>();
        }

        #endregion
    }
}
```

RssItem.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Collections.Generic;

namespace Aurora.Feed.Types.Rss
{
    public class RssItem
    {
        #region Attributes

        private string guid;
        private string title;
        private string link;
```

```
private string description;
private string author;
private Nullable<DateTime> pubDate;
private List<RssCategory> categories;
private ItemSource source;
private ItemEnclosure enclosure;
private string comments;

#endregion

#region Properties

public string Guid
{
    get { return this.guid; }
    set { this.guid = value; }
}

public string Title
{
    get { return this.title; }
    set { this.title = value; }
}

public string Link
{
    get { return this.link; }
    set { this.link = value; }
}

public string Description
{
    get { return this.description; }
    set { this.description = value; }
}

public string Author
{
    get { return this.author; }
    set { this.author = value; }
}

public Nullable<DateTime> PubDate
{
    get { return this.pubDate; }
    set { this.pubDate = value; }
}

public List<RssCategory> Categories
{
    get { return this.categories; }
    set { this.categories = value; }
}

public ItemSource Source
{
    get { return this.source; }
    set { this.source = value; }
}

public ItemEnclosure Enclosure
{
    get { return this.enclosure; }
```

```
            set { this.enclosure = value; }
        }

        public string Comments
        {
            get { return this.comments; }
            set { this.comments = value; }
        }

        #endregion

        #region Constructor

        public RssItem()
        {
            categories = new List<RssCategory>();
        }

        #endregion
    }
}
```

RssCategory.cs

```
using System;
using System.Data;
using System.Configuration;

namespace Aurora.Feed.Types.Rss
{
    public class RssCategory
    {
        #region Attributes

        private string domain;
        private string value;

        #endregion

        #region Properties

        public string Domain
        {
            get { return this.domain; }
            set { this.domain = value; }
        }

        public string Value
        {
            get { return this.value; }
            set { this.value = value; }
        }

        #endregion

        #region Constructor

        public RssCategory()
        {
            //construtor da classe RssCategory
        }
```

```
        #endregion
    }
}
```

ItemSource.cs

```
using System;
using System.Data;
using System.Configuration;

namespace Aurora.Feed.Types.Rss
{
    public class ItemSource
    {
        #region Attributes

        private string url;
        private string value;

        #endregion

        #region Properties

        public string Url
        {
            get { return this.url; }
            set { this.url = value; }
        }

        public string Value
        {
            get { return this.value; }
            set { this.value = value; }
        }

        #endregion

        #region Constructor

        public ItemSource()
        {
            //construtor da classe ItemSource
        }

        #endregion
    }
}
```

ItemEnclosure.cs

```
using System;
using System.Data;
using System.Configuration;

namespace Aurora.Feed.Types.Rss
{
    public class ItemEnclosure
    {
        #region Attributes
```

```
        private string url;
        private string type;
        private long length;

        #endregion

        #region Properties

        public string Url
        {
            get { return this.url; }
            set { this.url = value; }
        }

        public string Type
        {
            get { return this.type; }
            set { this.type = value; }
        }

        public long Length
        {
            get { return this.length; }
            set { this.length = value; }
        }

        #endregion

        #region Constructor

        public ItemEnclosure()
        {
            //construtor da classe ItemEnclosure;
        }

        #endregion
    }
}
```

ChannelImage.cs

```
using System;
using System.Data;
using System.Configuration;

namespace Aurora.Feed.Types.Rss
{
    public class ChannelImage
    {
        #region Attributes

        private string title;
        private string link;
        private string url;
        private string description;
        private int width;
        private int height;

        #endregion

        #region Properties
```

```
        public string Title
        {
            get { return this.title; }
            set { this.title = value; }
        }

        public string Link
        {
            get { return this.link; }
            set { this.link = value; }
        }

        public string Url
        {
            get { return this.url; }
            set { this.url = value; }
        }

        public string Description
        {
            get { return this.description; }
            set { this.description = value; }
        }

        public int Width
        {
            get { return this.width; }
            set { this.width = value; }
        }

        public int Height
        {
            get { return this.height; }
            set { this.height = value; }
        }

        #endregion

        #region Constructor

        public ChannelImage()
        {
            this.width = 88;
            this.height = 31;
        }

        #endregion
    }
}
```

ChannelCloud.cs

```
using System;
using System.Data;
using System.Configuration;

namespace Aurora.Feed.Types.Rss
{
    public class ChannelCloud
    {
        #region Attributes
```

```
        private string domain;
        private int port;
        private string path;
        private string registerProcedure;
        private string protocol;

        #endregion

        #region Properties

        public string Domain
        {
            get { return this.domain; }
            set { this.domain = value; }
        }

        public int Port
        {
            get { return this.port; }
            set { this.port = value; }
        }

        public string Path
        {
            get { return this.path; }
            set { this.path = value; }
        }

        public string RegisterProcedure
        {
            get { return this.registerProcedure; }
            set { this.registerProcedure = value; }
        }

        public string Protocol
        {
            get { return this.protocol; }
            set { this.protocol = value; }
        }

        #endregion

        #region Constructor

        public ChannelCloud()
        {
            //construtor da classe ChannelCloud;
        }

        #endregion
    }
}
```

AtomFeed.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Collections.Generic;
using System.Globalization;

namespace Aurora.Feed.Types.Atom
```

```
{
    /// <summary>
    /// Classe que armazena os dados extraídos através do parser para o feed Atom.
    /// Suporta a versão 1.0 do formato Atom.
    /// </summary>
    public class AtomFeed : HardwiredFeed
    {
        #region Attributes

        private string id;
        private AtomTextConstruct title;
        private Nullable<DateTime> updated;
        private List<AtomPersonConstruct> authors;
        private List<FeedLink> links;
        private List<FeedCategory> categories;
        private List<AtomPersonConstruct> contributors;
        private FeedGenerator generator;
        private string icon;
        private string logo;
        private AtomTextConstruct rights;
        private AtomTextConstruct subtitle;
        private List<AtomEntry> entries;
        private CultureInfo language;

        #endregion

        #region Properties

        public string Id
        {
            get { return this.id; }
            set { this.id = value; }
        }

        public AtomTextConstruct Title
        {
            get { return this.title; }
            set { this.title = value; }
        }

        public Nullable<DateTime> Updated
        {
            get { return this.updated; }
            set { this.updated = value; }
        }

        public List<AtomPersonConstruct> Authors
        {
            get { return this.authors; }
            set { this.authors = value; }
        }

        public List<FeedLink> Links
        {
            get { return this.links; }
            set { this.links = value; }
        }

        public List<FeedCategory> Categories
        {
            get { return this.categories; }
            set { this.categories = value; }
        }
```

```
public List<AtomPersonConstruct> Contributors
{
    get { return this.contributors; }
    set { this.contributors = value; }
}

public FeedGenerator Generator
{
    get { return this.generator; }
    set { this.generator = value; }
}

public string Icon
{
    get { return this.icon; }
    set { this.icon = value; }
}

public string Logo
{
    get { return this.logo; }
    set { this.logo = value; }
}

public AtomTextConstruct Rights
{
    get { return this.rights; }
    set { this.rights = value; }
}

public AtomTextConstruct Subtitle
{
    get { return this.subtitle; }
    set { this.subtitle = value; }
}

public List<AtomEntry> Entries
{
    get { return this.entries; }
    set { this.entries = value; }
}

public CultureInfo Language
{
    get { return this.language; }
    set { this.language = value; }
}

#endregion

#region Constructor

public AtomFeed()
{
    authors = new List<AtomPersonConstruct>();
    links = new List<FeedLink>();
    categories = new List<FeedCategory>();
    contributors = new List<AtomPersonConstruct>();
    entries = new List<AtomEntry>();
}

#endregion
```

```
    }
}
```

AtomEntry.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Collections.Generic;
using System.Globalization;

namespace Aurora.Feed.Types.Atom
{
    public class AtomEntry
    {
        #region Attributes

        private string id;
        private AtomTextConstruct title;
        private Nullable<DateTime> updated;
        private List<AtomPersonConstruct> authors;
        private EntryContent content;
        private AtomTextConstruct summary;
        private List<FeedLink> links;
        private List<FeedCategory> categories;
        private List<AtomPersonConstruct> contributors;
        private Nullable<DateTime> published;
        private EntrySource source;
        private AtomTextConstruct rights;

        #endregion

        #region Properties

        public string Id
        {
            get { return this.id; }
            set { this.id = value; }
        }

        public AtomTextConstruct Title
        {
            get { return this.title; }
            set { this.title = value; }
        }

        public Nullable<DateTime> Updated
        {
            get { return this.updated; }
            set { this.updated = value; }
        }

        public List<AtomPersonConstruct> Authors
        {
            get { return this.authors; }
            set { this.authors = value; }
        }

        public EntryContent Content
        {
            get { return this.content; }
            set { this.content = value; }
```

```
        }

        public AtomTextConstruct Summary
        {
            get { return this.summary; }
            set { this.summary = value; }
        }

        public List<FeedLink> Links
        {
            get { return this.links; }
            set { this.links = value; }
        }

        public List<FeedCategory> Categories
        {
            get { return this.categories; }
            set { this.categories = value; }
        }

        public List<AtomPersonConstruct> Contributors
        {
            get { return this.contributors; }
            set { this.contributors = value; }
        }

        public Nullable<DateTime> Published
        {
            get { return this.published; }
            set { this.published = value; }
        }

        public EntrySource Source
        {
            get { return this.source; }
            set { this.source = value; }
        }

        public AtomTextConstruct Rights
        {
            get { return this.rights; }
            set { this.rights = value; }
        }

        #endregion

        #region Constructor

        public AtomEntry()
        {
            authors = new List<AtomPersonConstruct>();
            links = new List<FeedLink>();
            categories = new List<FeedCategory>();
            contributors = new List<AtomPersonConstruct>();
        }

        #endregion
    }
}
```

AtomTextConstruct.cs

```
using System;
using System.Data;
using System.Configuration;

namespace Aurora.Feed.Types.Atom
{
    public class AtomTextConstruct
    {
        #region Attributes

        private string type;
        private string value;

        #endregion

        #region Properties

        public string Type
        {
            get { return this.type; }
            set { this.type = value; }
        }

        public string Value
        {
            get { return this.value; }
            set { this.value = value; }
        }

        #endregion

        #region Constructor

        public AtomTextConstruct()
        {
            Type = "text";
        }

        #endregion
    }
}
```

AtomPersonConstruct.cs

```
using System;
using System.Data;
using System.Configuration;

namespace Aurora.Feed.Types.Atom
{
    public class AtomPersonConstruct
    {
        #region Attributes

        private string name;
        private string uri;
        private string email;

        #endregion
```

```
        #region Properties

        public string Name
        {
            get { return this.name; }
            set { this.name = value; }
        }

        public string Uri
        {
            get { return this.uri; }
            set { this.uri = value; }
        }

        public string Email
        {
            get { return this.email; }
            set { this.email = value; }
        }

        #endregion

        #region Constructor

        public AtomPersonConstruct()
        {
            // construtor da classe AtomPersonConstruct
        }

        #endregion

    }
}
```

FeedCategory.cs

```
using System;
using System.Data;
using System.Configuration;

namespace Aurora.Feed.Types.Atom
{
    public class FeedCategory
    {
        #region Attributes

        private string term;
        private string scheme;
        private string label;

        #endregion

        #region Properties

        public string Term
        {
            get { return this.term; }
            set { this.term = value; }
        }

        public string Scheme
        {
```

```
            get { return this.scheme; }
            set { this.scheme = value; }
        }

        public string Label
        {
            get { return this.label; }
            set { this.label = value; }
        }

        #endregion

        #region Constructor

        public FeedCategory()
        {
            //construtor da classe FeedCategory
        }

        #endregion
    }
}
```

FeedGenerator.cs

```
using System;
using System.Data;
using System.Configuration;

namespace Aurora.Feed.Types.Atom
{
    public class FeedGenerator
    {
        #region Attributes

        private string value;
        private string uri;
        private string version;

        #endregion

        #region Properties

        public string Value
        {
            get { return this.value; }
            set { this.value = value; }
        }

        public string Uri
        {
            get { return this.uri; }
            set { this.uri = value; }
        }

        public string Version
        {
            get { return this.version; }
            set { this.version = value; }
        }

        #endregion
```

```
        #region Constructor

        public FeedGenerator()
        {
            //construtor da classe FeedGenerator
        }

        #endregion
    }
}
```

FeedLink.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Collections.Generic;
using System.Globalization;


namespace Aurora.Feed.Types.Atom
{
    public class FeedLink
    {
        #region Attributes

        private string href;
        private string rel;
        private string type;
        private CultureInfo hreflang;
        private string title;
        private long length;

        #endregion

        #region Properties

        public string Href
        {
            get { return this.href; }
            set { this.href = value; }
        }

        public string Rel
        {
            get { return this.rel; }
            set { this.rel = value; }
        }

        public string Type
        {
            get { return this.type; }
            set { this.type = value; }
        }

        public CultureInfo Hreflang
        {
            get { return this.hreflang; }
            set { this.hreflang = value; }
        }
```

```
        public string Title
        {
            get { return this.title; }
            set { this.title = value; }
        }

        public long Length
        {
            get { return this.length; }
            set { this.length = value; }
        }

        #endregion

        #region Constructor

        public FeedLink()
        {
            Rel = "alternate";
        }

        #endregion
    }
}
```

EntryContent.cs

```
using System;
using System.Data;
using System.Configuration;

namespace Aurora.Feed.Types.Atom
{
    public class EntryContent
    {
        #region Attributes

        private string type;
        private string source;
        private string value;

        #endregion

        #region Properties

        public string Type
        {
            get { return this.type; }
            set { this.type = value; }
        }

        public string Source
        {
            get { return this.source; }
            set { this.source = value; }
        }

        public string Value
        {
            get { return this.value; }
            set { this.value = value; }
        }
```

```
        #endregion

        #region Constructor

        public EntryContent()
        {
            //construtor da classe EntryContent
        }

        #endregion
    }
}
```

EntrySource.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Collections.Generic;
using System.Globalization;

namespace Aurora.Feed.Types.Atom
{
    public class EntrySource
    {
        #region Attributes

        private string id;
        private AtomTextConstruct title;
        private Nullable<DateTime> updated;
        private List<AtomPersonConstruct> authors;
        private List<FeedLink> links;
        private List<FeedCategory> categories;
        private List<AtomPersonConstruct> contributors;
        private FeedGenerator generator;
        private string icon;
        private string logo;
        private AtomTextConstruct rights;
        private AtomTextConstruct subtitle;

        #endregion

        #region Properties

        public string Id
        {
            get { return this.id; }
            set { this.id = value; }
        }

        public AtomTextConstruct Title
        {
            get { return this.title; }
            set { this.title = value; }
        }

        public Nullable<DateTime> Updated
        {
            get { return this.updated; }
            set { this.updated = value; }
        }
```

```
public List<AtomPersonConstruct> Authors
{
    get { return this.authors; }
    set { this.authors = value; }
}

public List<FeedLink> Links
{
    get { return this.links; }
    set { this.links = value; }
}

public List<FeedCategory> Categories
{
    get { return this.categories; }
    set { this.categories = value; }
}

public List<AtomPersonConstruct> Contributors
{
    get { return this.contributors; }
    set { this.contributors = value; }
}

public FeedGenerator Generator
{
    get { return this.generator; }
    set { this.generator = value; }
}

public string Icon
{
    get { return this.icon; }
    set { this.icon = value; }
}

public string Logo
{
    get { return this.logo; }
    set { this.logo = value; }
}

public AtomTextConstruct Rights
{
    get { return this.rights; }
    set { this.rights = value; }
}

public AtomTextConstruct Subtitle
{
    get { return this.subtitle; }
    set { this.subtitle = value; }
}

#endregion

#region Constructor

public EntrySource()
{
    authors = new List<AtomPersonConstruct>();
    links = new List<FeedLink>();
```

```
            categories = new List<FeedCategory>();
            contributors = new List<AtomPersonConstruct>();
        }

        #endregion
    }
}
```

NewsFeed.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Collections.Generic;
using System.Globalization;

namespace Aurora.Feed.Types
{
    public class NewsFeed
    {
        #region Attributes

        private string feedUrl;
        private string title;
        private NewsLink link;
        private string description;
        private CultureInfo language;
        private string image;
        private string copyright;
        private string generator;
        private Nullable<DateTime> updated;
        private string author;
        private List<NewsCategory> categories;
        private List<NewsItem> items;

        #endregion

        #region Properties

        public string FeedUrl
        {
            get { return this.feedUrl; }
            set { this.feedUrl = value; }
        }

        public string Title
        {
            get { return this.title; }
            set { this.title = value; }
        }

        public string Description
        {
            get { return this.description; }
            set { this.description = value; }
        }

        public NewsLink Link
        {
            get { return this.link; }
            set { this.link = value; }
        }
```

```
        public CultureInfo Language
        {
            get { return this.language; }
            set { this.language = value; }
        }

        public string Image
        {
            get { return this.image; }
            set { this.image = value; }
        }

        public string Copyright
        {
            get { return this.copyright; }
            set { this.copyright = value; }
        }

        public string Generator
        {
            get { return this.generator; }
            set { this.generator = value; }
        }

        public Nullable<DateTime> Updated
        {
            get { return this.updated; }
            set { this.updated = value; }
        }

        public string Author
        {
            get { return this.author; }
            set { this.author = value; }
        }

        public List<NewsCategory> Categories
        {
            get { return this.categories; }
            set { this.categories = value; }
        }

        public List<NewsItem> Items
        {
            get { return this.items; }
            set { this.items = value; }
        }

        #endregion

        #region Constructor

        public NewsFeed()
        {
            categories = new List<NewsCategory>();
            items = new List<NewsItem>();
        }

        public NewsFeed( string feedUrl, string title, string link, string description, string
language, string imageUrl, string copyright, string generator, DateTime updated, string author
)
        {
```

```
            this.feedUrl = feedUrl;
            this.title = title;
            this.description = description;
            this.link = new NewsLink( link );
            this.language = CultureInfo.GetCultureInfo( language );
            this.image = imageUrl;
            this.copyright = copyright;
            this.generator = generator;
            this.updated = updated;
            this.author = author;
            categories = new List<NewsCategory>();
            items = new List<NewsItem>();
        }

        #endregion
    }
}
```

NewsItem.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Collections.Generic;
using System.Globalization;

namespace Aurora.Feed.Types
{

    public class NewsItem
    {
        #region Attributes

        private string itemUrl;
        private string feedUrl;
        private string title;
        private string content;
        private NewsLink link;
        private string author;
        private Nullable<DateTime> published;
        private List<NewsCategory> categories;
        private NewsLink source;
        private NewsLink enclosure;
        private List<NewsItemKeyword> keywords;
        private bool isReaded;
        private float interestingness;

        #endregion

        #region Properties

        public string ItemUrl
        {
            get { return this.itemUrl; }
            set { this.itemUrl = value; }
        }

        public string FeedUrl
        {
            get { return this.feedUrl; }
            set { this.feedUrl = value; }
        }
```

```
public string Title
{
    get { return this.title; }
    set { this.title = value; }
}

public string Content
{
    get { return this.content; }
    set { this.content = value; }
}

public NewsLink Link
{
    get { return this.link; }
    set { this.link = value; }
}

public string Author
{
    get { return this.author; }
    set { this.author = value; }
}

public Nullable<DateTime> Published
{
    get { return this.published; }
    set { this.published = value; }
}

public List<NewsCategory> Categories
{
    get { return this.categories; }
    set { this.categories = value; }
}

public NewsLink Source
{
    get { return this.source; }
    set { this.source = value; }
}

public NewsLink Enclosure
{
    get { return this.enclosure; }
    set { this.enclosure = value; }
}

public List<NewsItemKeyword> Keywords
{
    get { return this.keywords; }
    set { this.keywords = value; }
}

public bool IsReaded
{
    get { return this.isReaded; }
    set { this.isReaded = value; }
}

public float Interestingness
{
```

```
            get { return this.interestingness; }
            set { this.interestingness = value; }
        }

        #endregion

        #region Constructor

        public NewsItem()
        {
            categories = new List<NewsCategory>();
            keywords = new List<NewsItemKeyword>();
            isReaded = false;
        }

        #endregion

        #region Methods

        public void AddKeyword( string value )
        {
            int i = 0;
            bool exists = false;

            for ( i = 0; i < keywords.Count; i++ )
            {
                if ( keywords[ i ].Value == value )
                {
                    exists = true;
                    break;
                }
            }

            if ( exists )
                keywords[ i ].Occurrences = keywords[ i ].Occurrences + 1;
            else
            {
                NewsItemKeyword keyword = new NewsItemKeyword( value, this.feedUrl,
this.itemUrl );
                keywords.Add( keyword );
            }
        }

        #endregion
    }
}
```

NewsCategory.cs

```
using System;
using System.Data;
using System.Configuration;

namespace Aurora.Feed.Types
{
    public class NewsCategory
    {
        #region Attributes

        private string term;
        private string domain;
```

```
        #endregion

        #region Properties

        public string Term
        {
            get { return this.term; }
            set { this.term = value; }
        }

        public string Domain
        {
            get { return this.domain; }
            set { this.domain = value; }
        }

        #endregion

        #region Constructor

        public NewsCategory()
        {
            //construtor da classe NewsCategory.
        }

        #endregion
    }
}
```

NewsLink.cs

```
using System;
using System.Data;
using System.Configuration;

namespace Aurora.Feed.Types
{
    public class NewsLink
    {
        #region Attributes

        private string url;
        private string title;
        private string type;
        private long length;

        #endregion

        #region Properties

        public string Url
        {
            get { return this.url; }
            set { this.url = value; }
        }

        public string Title
        {
            get { return this.title; }
            set { this.title = value; }
        }
```

```
        public string Type
        {
            get { return this.type; }
            set { this.type = value; }
        }

        public long Length
        {
            get { return this.length; }
            set { this.length = value; }
        }

        #endregion

        #region Constructor

        public NewsLink()
        {
            //construtor da classe NewsLink.
        }

        public NewsLink( string url )
        {
            this.url = url;
        }

        #endregion
    }
}
```

Keyword.cs

```
using System;
using System.Data;
using System.Configuration;

namespace Aurora.Feed.Types
{
    public abstract class Keyword
    {
        #region Attributes

        private string value;

        #endregion

        #region Properties

        public string Value
        {
            get { return this.value; }
            set { this.value = value; }
        }

        #endregion

        #region Constructor

        public Keyword()
        {
            // TODO: Add constructor logic here
        }
```

```
        #endregion
    }
}
```

NewsItemKeyword.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using Aurora.Feed.Types;


namespace Aurora.Feed.Types
{
    public class NewsItemKeyword : Keyword
    {
        #region Attributes

        private string feedUrl;
        private string itemUrl;
        private int occurrences;

        #endregion

        #region Properties

        public string FeedUrl
        {
            get { return this.feedUrl; }
            set { this.feedUrl = value; }
        }

        public string ItemUrl
        {
            get { return this.itemUrl; }
            set { this.itemUrl = value; }
        }

        public int Occurrences
        {
            get { return this.occurrences; }
            set { this.occurrences = value; }
        }

        #endregion

        #region Constructor

        public NewsItemKeyword( string value, string feedUrl, string itemUrl )
        {
            this.Value = value;
            this.Occurrences = 1;
            this.feedUrl = feedUrl;
            this.itemUrl = itemUrl;
        }

        public NewsItemKeyword( string value, string feedUrl, string itemUrl, int occurrences
)
        {
            this.Value = value;
            this.Occurrences = occurrences;
```

```
            this.feedUrl = feedUrl;
            this.itemUrl = itemUrl;
        }

        #endregion
    }
}
```

UserKeyword.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using Aurora.Feed.Types;

namespace Aurora.Types
{
    public class UserKeyword : Keyword
    {
        #region Attributes

        private string userId;
        private int interestingOccurrences;
        private int notInterestingOccurrences;
        private float interestingFrequency;
        private float notInterestingFrequency;

        #endregion

        #region Properties

        public string UserId
        {
            get { return this.userId; }
            set { this.userId = value; }
        }

        public int InterestingOccurrences
        {
            get { return this.interestingOccurrences; }
            set { this.interestingOccurrences = value; }
        }

        public int NotInterestingOccurrences
        {
            get { return this.notInterestingOccurrences; }
            set { this.notInterestingOccurrences = value; }
        }

        public float InterestingFrequency
        {
            get { return this.interestingFrequency; }
            set { this.interestingFrequency = value; }
        }

        public float NotInterestingFrequency
        {
            get { return this.notInterestingFrequency; }
            set { this.notInterestingFrequency = value; }
        }

        #endregion
```

```
        #region Constructor

        public UserKeyword( string value, string userId )
        {
            this.Value = value;
            this.userId = userId;
            this.interestingOccurrences = 0;
            this.notInterestingOccurrences = 0;
        }

        public UserKeyword( string value, string userId, int interestingOccurrences, int
notInterestingOccurrences )
        {
            this.Value = value;
            this.userId = userId;
            this.interestingOccurrences = interestingOccurrences;
            this.notInterestingOccurrences = notInterestingOccurrences;
        }

        #endregion
    }
}
```

FeedReader.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Xml;
using Aurora.Feed.Types;
using Aurora.Feed.Parsers;
using Aurora.Feed.Extractor;
using Aurora.Feed.Converters;

public class FeedReader
{
    public FeedReader()
    {
        //construtor
    }

    public NewsFeed Load( string url )
    {
        XmlDocument xmlDocument = new XmlDocument();
        xmlDocument.Load( url );
        HardwiredFeed hardwiredFeed = FeedParser.Parse( xmlDocument );
        NewsFeed newsFeed = FeedConverter.Convert( hardwiredFeed );
        KeywordExtractor.ExtractKeywords( newsFeed );
        return newsFeed;
    }
}
```

FeedParser.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Xml;
using Aurora.Feed.Types;
using Aurora.Feed.Parsers.Rss;
```

```
using Aurora.Feed.Parsers.Atom;


namespace Aurora.Feed.Parsers
{
    public class FeedParser
    {
        public FeedParser()
        {
            //construtor
        }

        public static HardwiredFeed Parse( XmlDocument xmlDocument )
        {
            XmlElement root = xmlDocument.DocumentElement;

            if ( root.Name == "rss" )
            {
                if ( root.Attributes.Count > 0 )
                {
                    XmlAttribute rssVersion = root.Attributes[ "version" ];

                    if ( null != rssVersion )
                    {
                        string version = rssVersion.Value;

                        if ( version == "0.91" )
                        {
                            Rss091Parser feedParser = new Rss091Parser();
                            return feedParser.Parse( xmlDocument );
                        }

                        if ( version == "0.92" )
                        {
                            Rss092Parser feedParser = new Rss092Parser();
                            return feedParser.Parse( xmlDocument );
                        }

                        if ( ( version == "2.0" ) || ( version == "2.00" ) )
                        {
                            Rss20Parser feedParser = new Rss20Parser();
                            return feedParser.Parse( xmlDocument );
                        }

                        return null; //a versão do RSS especificada no XML não é suportada
pela aplicação
                    }
                }

                return null; //não foi possível identificar a versão do RSS utilizada no XML
            }

            if ( ( root.Name == "feed" ) || ( root.Name == "rdf:RDF" ) )
            {
                if ( root.Attributes.Count > 0 )
                {
                    XmlAttribute feedVersion = root.Attributes[ "xmlns" ];
                    if ( null != feedVersion )
                    {
                        if ( feedVersion.Value == "http://www.w3.org/2005/Atom" )
                        {
                            Atom10Parser feedParser = new Atom10Parser();
                            return feedParser.Parse( xmlDocument );
```

```
                        }

                        if ( feedVersion.Value == "http://purl.org/rss/1.0/" )
                        {
                            Rss10Parser feedParser = new Rss10Parser();
                            return feedParser.Parse( xmlDocument );
                        }

                        return null; //a versão do feed especificada no XML não é suportada
pela aplicação
                    }
                }

                return null; //não foi possível identificar a versão do feed utilizada no XML
            }

            return null; //não foi possível identificar o tipo do feed especificado no XML
        }
    }
}
```

Rss091Parser.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Xml;
using System.Collections.Generic;
using System.Globalization;
using Aurora.Feed.Types.Rss;

namespace Aurora.Feed.Parsers.Rss
{
    public class Rss091Parser
    {
        #region Constructor

        public Rss091Parser()
        {
            //construtor da classe Rss091Parser
        }

        #endregion

        #region Parser

        public RssChannel Parse( XmlDocument xmlDocument )
        {
            RssChannel rssChannel = new RssChannel();

            rssChannel.FeedUrl = xmlDocument.BaseURI;
            rssChannel.FeedType = FeedType.RSS091;

            if ( xmlDocument.FirstChild.NodeType == XmlNodeType.XmlDeclaration )
            {
                XmlDeclaration xmlDeclaration = ( XmlDeclaration )xmlDocument.FirstChild;
                rssChannel.XmlVersion = xmlDeclaration.Version;
                rssChannel.XmlEncoding = xmlDeclaration.Encoding;
            }

            XmlElement root = xmlDocument.DocumentElement;
```

```
    foreach ( XmlNode node in root.FirstChild.ChildNodes )
    {
        if ( node.Name == "title" )
            rssChannel.Title = node.InnerText;

        if ( node.Name == "link" )
            rssChannel.Link = node.InnerText;

        if ( node.Name == "description" )
            rssChannel.Description = node.InnerText;

        if ( node.Name == "language" )
            rssChannel.Language = ParseLanguage( node );

        if ( node.Name == "image" )
            rssChannel.Image = ParseImage( node );

        if ( node.Name == "copyright" )
            rssChannel.Copyright = node.InnerText;

        if ( node.Name == "lastBuildDate" )
            rssChannel.LastBuildDate = ParseDate( node );

        if ( node.Name == "pubDate" )
            rssChannel.PubDate = ParseDate( node );

        if ( node.Name == "managingEditor" )
            rssChannel.ManagingEditor = node.InnerText;

        if ( node.Name == "item" )
            rssChannel.Items.Add( ParseItem( node ) );

        if ( node.Name == "webMaster" )
            rssChannel.WebMaster = node.InnerText;

        if ( node.Name == "rating" )
            rssChannel.Rating = node.InnerText;

        if ( node.Name == "docs" )
            rssChannel.Docs = node.InnerText;

        if ( node.Name == "skipHours" )
            rssChannel.SkipHours = ParseSkipHours( node );

        if ( node.Name == "skipDays" )
            rssChannel.SkipDays = ParseSkipDays( node );
    }

    return rssChannel;
}

private CultureInfo ParseLanguage( XmlNode node )
{
    try
    {
        return System.Globalization.CultureInfo.GetCultureInfo( node.InnerText );
    }
    catch
    {
        /*gravar log.*/
    }
```

```
            return null;  //é executado somente quando ocorre um erro ao obter a informação de
localização
        }

        private ChannelImage ParseImage( XmlNode node )
        {
            ChannelImage channelImage = new ChannelImage();

            foreach ( XmlNode imageNode in node.ChildNodes )
            {
                if ( imageNode.Name == "title" )
                    channelImage.Title = imageNode.InnerText;

                if ( imageNode.Name == "link" )
                    channelImage.Link = imageNode.InnerText;

                if ( imageNode.Name == "url" )
                    channelImage.Url = imageNode.InnerText;

                if ( imageNode.Name == "description" )
                    channelImage.Description = imageNode.InnerText;

                if ( imageNode.Name == "width" )
                {
                    try
                    {
                        channelImage.Width = int.Parse( imageNode.InnerText );
                    }
                    catch
                    {
                        /*gravar log*/
                    }
                }

                if ( imageNode.Name == "height" )
                {
                    try
                    {
                        channelImage.Height = int.Parse( imageNode.InnerText );
                    }
                    catch
                    {
                        /*gravar log*/
                    }
                }
            }

            return channelImage;
        }

        private Nullable<DateTime> ParseDate( XmlNode node )
        {
            try
            {
                return DateTime.Parse( node.InnerText );
            }
            catch
            {
                /*gravar log*/
            }

            return null;
        }
```

```
        private RssItem ParseItem( XmlNode node )
        {
            RssItem rssItem = new RssItem();

            foreach ( XmlNode itemNode in node.ChildNodes )
            {
                if ( itemNode.Name == "title" )
                    rssItem.Title = itemNode.InnerText;

                if ( itemNode.Name == "link" )
                    rssItem.Link = itemNode.InnerText;

                if ( itemNode.Name == "description" )
                    rssItem.Description = itemNode.InnerText;
            }

            return rssItem;
        }

        private int ParseInt( XmlNode node )
        {
            try
            {
                return int.Parse( node.InnerText );
            }
            catch
            {
                /*gravar log*/
            }

            return 0;
        }

        private List<int> ParseSkipHours( XmlNode node )
        {
            List<int> skipHours = new List<int>();

            foreach ( XmlNode itemNode in node.ChildNodes )
            {
                if ( itemNode.Name == "hour" )
                    skipHours.Add( ParseInt( itemNode ) );
            }

            return skipHours;
        }

        private List<DayOfWeek> ParseSkipDays( XmlNode node )
        {
            List<DayOfWeek> skipDays = new List<DayOfWeek>();

            //TODO: testar se o dia vier em formato errado

            foreach ( XmlNode itemNode in node.ChildNodes )
            {
                if ( itemNode.Name == "day" )
                {
                    DayOfWeek dayOfWeek = ( DayOfWeek )Enum.Parse( typeof( DayOfWeek ),
itemNode.InnerText, true );
                    skipDays.Add( dayOfWeek );
                }
            }
```

```
            return skipDays;
        }

        #endregion
    }
}
```

Rss092Parser.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Xml;
using System.Collections.Generic;
using System.Globalization;
using Aurora.Feed.Types.Rss;

namespace Aurora.Feed.Parsers.Rss
{
    public class Rss092Parser
    {
        #region Constructor

        public Rss092Parser()
        {
            //construtor da classe Rss092Parser
        }

        #endregion

        #region Parser

        public RssChannel Parse( XmlDocument xmlDocument )
        {
            RssChannel rssChannel = new RssChannel();

            rssChannel.FeedUrl = xmlDocument.BaseURI;
            rssChannel.FeedType = FeedType.RSS092;

            if ( xmlDocument.FirstChild.NodeType == XmlNodeType.XmlDeclaration )
            {
                XmlDeclaration xmlDeclaration = ( XmlDeclaration )xmlDocument.FirstChild;
                rssChannel.XmlVersion = xmlDeclaration.Version;
                rssChannel.XmlEncoding = xmlDeclaration.Encoding;
            }

            XmlElement root = xmlDocument.DocumentElement;

            foreach ( XmlNode node in root.FirstChild.ChildNodes )
            {
                if ( node.Name == "title" )
                    rssChannel.Title = node.InnerText;

                if ( node.Name == "link" )
                    rssChannel.Title = node.InnerText;

                if ( node.Name == "description" )
                    rssChannel.Description = node.InnerText;

                if ( node.Name == "image" )
                    rssChannel.Image = ParseImage( node );
```

```
        if ( node.Name == "language" )
            rssChannel.Language = ParseLanguage( node );

        if ( node.Name == "copyright" )
            rssChannel.Copyright = node.InnerText;

        if ( node.Name == "lastBuildDate" )
            rssChannel.LastBuildDate = ParseDate( node );

        if ( node.Name == "pubDate" )
            rssChannel.PubDate = ParseDate( node );

        if ( node.Name == "managingEditor" )
            rssChannel.ManagingEditor = node.InnerText;

        if ( node.Name == "item" )
            rssChannel.Items.Add( ParseItem( node ) );

        if ( node.Name == "webMaster" )
            rssChannel.WebMaster = node.InnerText;

        if ( node.Name == "rating" )
            rssChannel.Rating = node.InnerText;

        if ( node.Name == "cloud" )
            rssChannel.Cloud = ParseCloud( node );

        if ( node.Name == "docs" )
            rssChannel.Docs = node.InnerText;

        if ( node.Name == "skipHours" )
            rssChannel.SkipHours = ParseSkipHours( node );

        if ( node.Name == "skipDays" )
            rssChannel.SkipDays = ParseSkipDays( node );
    }

    return rssChannel;
}

private ChannelImage ParseImage( XmlNode node )
{
    ChannelImage channelImage = new ChannelImage();

    foreach ( XmlNode imageNode in node.ChildNodes )
    {
        if ( imageNode.Name == "title" )
            channelImage.Title = imageNode.InnerText;

        if ( imageNode.Name == "link" )
            channelImage.Link = imageNode.InnerText;

        if ( imageNode.Name == "url" )
            channelImage.Url = imageNode.InnerText;

        if ( imageNode.Name == "description" )
            channelImage.Description = imageNode.InnerText;

        if ( imageNode.Name == "width" )
        {
            try
            {
                channelImage.Width = int.Parse( imageNode.InnerText );
```

```
                }
                catch
                {
                    /*gravar log*/
                }
            }

            if ( imageNode.Name == "height" )
            {
                try
                {
                    channelImage.Height = int.Parse( imageNode.InnerText );
                }
                catch
                {
                    /*gravar log*/
                }
            }
        }

        return channelImage;
    }

    private CultureInfo ParseLanguage( XmlNode node )
    {
        try
        {
            return System.Globalization.CultureInfo.GetCultureInfo( node.InnerText );
        }
        catch
        {
            /*gravar log.*/
        }

        return null;  //é executado somente quando ocorre um erro ao obter a informação de
localização
    }

    private Nullable<DateTime> ParseDate( XmlNode node )
    {
        try
        {
            return DateTime.Parse( node.InnerText );
        }
        catch
        {
            /*gravar log*/
        }

        return null;
    }

    private RssItem ParseItem( XmlNode node )
    {
        RssItem rssItem = new RssItem();

        foreach ( XmlNode itemNode in node.ChildNodes )
        {
            if ( itemNode.Name == "title" )
                rssItem.Title = itemNode.InnerText;

            if ( itemNode.Name == "link" )
                rssItem.Link = itemNode.InnerText;
```

```
        if ( itemNode.Name == "description" )
            rssItem.Description = itemNode.InnerText;

        if ( itemNode.Name == "category" )
            rssItem.Categories.Add( ParseCategory( itemNode ) );

        if ( itemNode.Name == "source" )
            rssItem.Source = ParseSource( itemNode );

        if ( itemNode.Name == "enclosure" )
            rssItem.Enclosure = ParseEnclosure( itemNode );
    }

    return rssItem;
}

private int ParseInt( XmlNode node )
{
    try
    {
        return int.Parse( node.InnerText );
    }
    catch
    {
        /*gravar log*/
    }

    return 0;
}

private RssCategory ParseCategory( XmlNode node )
{
    RssCategory rssCategory = new RssCategory();

    if ( node.Attributes.Count > 0 )
    {
        XmlAttribute attribute = node.Attributes[ "domain" ];
        if ( null != attribute )
            rssCategory.Domain = attribute.Value;
    }

    rssCategory.Value = node.InnerText;

    return rssCategory;
}

private ItemSource ParseSource( XmlNode node )
{
    ItemSource itemSource = new ItemSource();

    if ( node.Attributes.Count > 0 )
    {
        XmlAttribute attribute = node.Attributes[ "url" ];
        if ( null != attribute )
            itemSource.Url = attribute.Value;
    }

    itemSource.Value = node.InnerText;

    return itemSource;
}
```

```
private ItemEnclosure ParseEnclosure( XmlNode node )
{
    ItemEnclosure itemEnclosure = new ItemEnclosure();

    if ( node.Attributes.Count > 0 )
    {
        XmlAttribute attribute = node.Attributes[ "url" ];
        if ( null != attribute )
            itemEnclosure.Url = attribute.Value;

        attribute = node.Attributes[ "length" ];
        if ( null != attribute )
        {
            try
            {
                itemEnclosure.Length = long.Parse( attribute.Value );
            }
            catch
            {
                /*gravar log*/
            }
        }

        attribute = node.Attributes[ "type" ];
        if ( null != attribute )
            itemEnclosure.Type = attribute.Value;
    }

    return itemEnclosure;
}

private ChannelCloud ParseCloud( XmlNode node )
{
    ChannelCloud channelCloud = new ChannelCloud();

    if ( node.Attributes.Count > 0 )
    {
        XmlAttribute attribute = node.Attributes[ "domain" ];
        if ( null != attribute )
            channelCloud.Domain = attribute.Value;


        attribute = node.Attributes[ "port" ];
        if ( null != attribute )
        {
            try
            {
                channelCloud.Port = int.Parse( attribute.Value );
            }
            catch
            {
                /*gravar log*/
            }

        }

        attribute = node.Attributes[ "path" ];
        if ( null != attribute )
            channelCloud.Path = attribute.Value;

        attribute = node.Attributes[ "registerProcedure" ];
        if ( null != attribute )
            channelCloud.RegisterProcedure = attribute.Value;
```

```
                attribute = node.Attributes[ "protocol" ];
                if ( null != attribute )
                    channelCloud.Protocol = attribute.Value;
            }

            return channelCloud;
        }

        private List<int> ParseSkipHours( XmlNode node )
        {
            List<int> skipHours = new List<int>();

            foreach ( XmlNode itemNode in node.ChildNodes )
            {
                if ( itemNode.Name == "hour" )
                    skipHours.Add( ParseInt( itemNode ) );
            }

            return skipHours;
        }

        private List<DayOfWeek> ParseSkipDays( XmlNode node )
        {
            List<DayOfWeek> skipDays = new List<DayOfWeek>();

            //TODO: testar se o dia vier em formato errado

            foreach ( XmlNode itemNode in node.ChildNodes )
            {
                if ( itemNode.Name == "day" )
                {
                    DayOfWeek dayOfWeek = ( DayOfWeek )Enum.Parse( typeof( DayOfWeek ),
itemNode.InnerText, true );
                    skipDays.Add( dayOfWeek );
                }
            }

            return skipDays;
        }

        #endregion
    }
}
```

Rss20Parser.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Xml;
using System.Collections.Generic;
using System.Globalization;
using Aurora.Feed.Types.Rss;

namespace Aurora.Feed.Parsers.Rss
{
    public class Rss20Parser
    {
        #region Constructor

        public Rss20Parser()
```

```
{
    //construtor da classe Rss20Parser
}

#endregion

#region Parser

public RssChannel Parse( XmlDocument xmlDocument )
{
    RssChannel rssChannel = new RssChannel();

    rssChannel.FeedUrl = xmlDocument.BaseURI;
    rssChannel.FeedType = FeedType.RSS20;

    if ( xmlDocument.FirstChild.NodeType == XmlNodeType.XmlDeclaration )
    {
        XmlDeclaration xmlDeclaration = ( XmlDeclaration )xmlDocument.FirstChild;
        rssChannel.XmlVersion = xmlDeclaration.Version;
        rssChannel.XmlEncoding = xmlDeclaration.Encoding;
    }

    XmlElement root = xmlDocument.DocumentElement;

    foreach ( XmlNode node in root.FirstChild.ChildNodes )
    {
        if ( node.Name == "title" )
            rssChannel.Title = node.InnerText;

        if ( node.Name == "link" )
            rssChannel.Link = node.InnerText;

        if ( node.Name == "description" )
            rssChannel.Description = node.InnerText;

        if ( node.Name == "language" )
            rssChannel.Language = ParseLanguage( node );

        if ( node.Name == "image" )
            rssChannel.Image = ParseImage( node );

        if ( node.Name == "copyright" )
            rssChannel.Copyright = node.InnerText;

        if ( node.Name == "generator" )
            rssChannel.Generator = node.InnerText;

        if ( node.Name == "lastBuildDate" )
            rssChannel.LastBuildDate = ParseDate( node );

        if ( node.Name == "pubDate" )
            rssChannel.PubDate = ParseDate( node );

        if ( node.Name == "managingEditor" )
            rssChannel.ManagingEditor = node.InnerText;

        if ( node.Name == "category" )
            rssChannel.Categories.Add( ParseCategory( node ) );

        if ( node.Name == "item" )
            rssChannel.Items.Add( ParseItem( node ) );

        if ( node.Name == "webMaster" )
```

```
                rssChannel.WebMaster = node.InnerText;

            if ( node.Name == "rating" )
                rssChannel.Rating = node.InnerText;

            if ( node.Name == "ttl" )
                rssChannel.TimeToLive = ParseInt( node );

            if ( node.Name == "cloud" )
                rssChannel.Cloud = ParseCloud( node );

            if ( node.Name == "docs" )
                rssChannel.Docs = node.InnerText;

            if ( node.Name == "skipHours" )
                rssChannel.SkipHours = ParseSkipHours( node );

            if ( node.Name == "skipDays" )
                rssChannel.SkipDays = ParseSkipDays( node );
        }

        return rssChannel;
    }

    private CultureInfo ParseLanguage( XmlNode node )
    {
        try
        {
            return System.Globalization.CultureInfo.GetCultureInfo( node.InnerText );
        }
        catch
        {
            /*gravar log.*/
        }

        return null;  //é executado somente quando ocorre um erro ao obter a informação de
localização
    }

    private Nullable<DateTime> ParseDate( XmlNode node )
    {
        try
        {
            return DateTime.Parse( node.InnerText );
        }
        catch
        {
            /*gravar log*/
        }

        return null;
    }

    private ChannelImage ParseImage( XmlNode node )
    {
        ChannelImage channelImage = new ChannelImage();

        foreach ( XmlNode imageNode in node.ChildNodes )
        {
            if ( imageNode.Name == "title" )
                channelImage.Title = imageNode.InnerText;

            if ( imageNode.Name == "link" )
```

```
            channelImage.Link = imageNode.InnerText;

        if ( imageNode.Name == "url" )
            channelImage.Url = imageNode.InnerText;

        if ( imageNode.Name == "description" )
            channelImage.Description = imageNode.InnerText;

        if ( imageNode.Name == "width" )
        {
            try
            {
                channelImage.Width = int.Parse( imageNode.InnerText );
            }
            catch
            {
                /*gravar log*/
            }
        }

        if ( imageNode.Name == "height" )
        {
            try
            {
                channelImage.Height = int.Parse( imageNode.InnerText );
            }
            catch
            {
                /*gravar log*/
            }
        }
    }

    return channelImage;
}

private RssCategory ParseCategory( XmlNode node )
{
    RssCategory rssCategory = new RssCategory();

    if ( node.Attributes.Count > 0 )
    {
        XmlAttribute attribute = node.Attributes[ "domain" ];
        if ( null != attribute )
            rssCategory.Domain = attribute.Value;
    }

    rssCategory.Value = node.InnerText;

    return rssCategory;
}

private RssItem ParseItem( XmlNode node )
{
    RssItem rssItem = new RssItem();

    foreach ( XmlNode itemNode in node.ChildNodes )
    {
        if ( itemNode.Name == "guid" )
            rssItem.Guid = itemNode.InnerText;

        if ( itemNode.Name == "title" )
            rssItem.Title = itemNode.InnerText;
```

```
        if ( itemNode.Name == "link" )
            rssItem.Link = itemNode.InnerText;

        if ( itemNode.Name == "description" )
            rssItem.Description = itemNode.InnerText;

        if ( itemNode.Name == "author" )
            rssItem.Author = itemNode.InnerText;

        if ( itemNode.Name == "pubDate" )
            rssItem.PubDate = ParseDate( itemNode );

        if ( itemNode.Name == "category" )
            rssItem.Categories.Add( ParseCategory( itemNode ) );

        if ( itemNode.Name == "source" )
            rssItem.Source = ParseSource( itemNode );

        if ( itemNode.Name == "enclosure" )
            rssItem.Enclosure = ParseEnclosure( itemNode );

        if ( itemNode.Name == "comments" )
            rssItem.Comments = itemNode.InnerText;
    }

    return rssItem;
}

private int ParseInt( XmlNode node )
{
    try
    {
        return int.Parse( node.InnerText );
    }
    catch
    {
        /*gravar log*/
    }

    return 0;
}

private ItemSource ParseSource( XmlNode node )
{
    ItemSource itemSource = new ItemSource();

    if ( node.Attributes.Count > 0 )
    {
        XmlAttribute attribute = node.Attributes[ "url" ];
        if ( null != attribute )
            itemSource.Url = attribute.Value;
    }

    itemSource.Value = node.InnerText;

    return itemSource;
}

private ItemEnclosure ParseEnclosure( XmlNode node )
{
    ItemEnclosure itemEnclosure = new ItemEnclosure();
```

```
    if ( node.Attributes.Count > 0 )
    {
        XmlAttribute attribute = node.Attributes[ "url" ];
        if ( null != attribute )
            itemEnclosure.Url = attribute.Value;

        attribute = node.Attributes[ "length" ];
        if ( null != attribute )
        {
            try
            {
                itemEnclosure.Length = long.Parse( attribute.Value );
            }
            catch
            {
                /*gravar log*/
            }
        }

        attribute = node.Attributes[ "type" ];
        if ( null != attribute )
            itemEnclosure.Type = attribute.Value;
    }

    return itemEnclosure;
}

private ChannelCloud ParseCloud( XmlNode node )
{
    ChannelCloud channelCloud = new ChannelCloud();

    if ( node.Attributes.Count > 0 )
    {
        XmlAttribute attribute = node.Attributes[ "domain" ];
        if ( null != attribute )
            channelCloud.Domain = attribute.Value;


        attribute = node.Attributes[ "port" ];
        if ( null != attribute )
        {
            try
            {
                channelCloud.Port = int.Parse( attribute.Value );
            }
            catch
            {
                /*gravar log*/
            }

        }

        attribute = node.Attributes[ "path" ];
        if ( null != attribute )
            channelCloud.Path = attribute.Value;

        attribute = node.Attributes[ "registerProcedure" ];
        if ( null != attribute )
            channelCloud.RegisterProcedure = attribute.Value;

        attribute = node.Attributes[ "protocol" ];
        if ( null != attribute )
            channelCloud.Protocol = attribute.Value;
```

```
            }

            return channelCloud;
        }

        private List<int> ParseSkipHours( XmlNode node )
        {
            List<int> skipHours = new List<int>();

            foreach ( XmlNode itemNode in node.ChildNodes )
            {
                if ( itemNode.Name == "hour" )
                    skipHours.Add( ParseInt( itemNode ) );
            }

            return skipHours;
        }

        private List<DayOfWeek> ParseSkipDays( XmlNode node )
        {
            List<DayOfWeek> skipDays = new List<DayOfWeek>();

            //TODO: testar se o dia vier em formato errado

            foreach ( XmlNode itemNode in node.ChildNodes )
            {
                if ( itemNode.Name == "day" )
                {
                    DayOfWeek dayOfWeek = ( DayOfWeek )Enum.Parse( typeof( DayOfWeek ),
itemNode.InnerText, true );
                    skipDays.Add( dayOfWeek );
                }
            }

            return skipDays;
        }

        #endregion
    }
}
```

Rss10Parser.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Xml;
using System.Collections.Generic;
using System.Globalization;
using Aurora.Feed.Types.Rss;


namespace Aurora.Feed.Parsers.Rss
{
    public class Rss10Parser
    {
        #region Constructor

        public Rss10Parser()
        {
            //construtor da classe Rss10Parser
        }
```

```
        #endregion

        #region Parser

        public RssChannel Parse( XmlDocument xmlDocument )
        {
            RssChannel rssChannel = new RssChannel();

            rssChannel.FeedUrl = xmlDocument.BaseURI;
            rssChannel.FeedType = FeedType.RSS10;

            if ( xmlDocument.FirstChild.NodeType == XmlNodeType.XmlDeclaration )
            {
                XmlDeclaration xmlDeclaration = ( XmlDeclaration )xmlDocument.FirstChild;
                rssChannel.XmlVersion = xmlDeclaration.Version;
                rssChannel.XmlEncoding = xmlDeclaration.Encoding;
            }

            XmlElement root = xmlDocument.DocumentElement;

            foreach ( XmlNode node in root.ChildNodes )
            {
                if ( node.Name == "channel" )   //o channel, mesmo sendo do mesmo nível
hierárquico do "image" e do "item", deve ter seu parse separado, "aberto" dentro do foreach do
"root.ChildNodes"
                {
                    foreach ( XmlNode channelNode in node.ChildNodes )
                    {
                        //TODO: parse do atributo rdf:about - realmente necessário?

                        if ( channelNode.Name == "title" )
                            rssChannel.Title = channelNode.InnerText;

                        if ( channelNode.Name == "link" )
                            rssChannel.Link = channelNode.InnerText;

                        if ( channelNode.Name == "description" )
                            rssChannel.Description = channelNode.InnerText;

                        if ( channelNode.Name == "dc:language" )
                            rssChannel.Language = ParseLanguage( channelNode );

                        if ( channelNode.Name == "dc:rights" )
                            rssChannel.Copyright = channelNode.InnerText;

                        if ( channelNode.Name == "dc:date" )
                            rssChannel.PubDate = ParseDate( channelNode );

                        if ( channelNode.Name == "dc:creator" )
                            rssChannel.ManagingEditor = channelNode.InnerText;

                        if ( channelNode.Name == "dc:subject" )
                            rssChannel.Categories.Add( ParseCategory( channelNode ) );

                        if ( channelNode.Name == "dc:publisher" )
                            rssChannel.WebMaster = channelNode.InnerText;
                    }
                }

                if ( node.Name == "image" )
                    rssChannel.Image = ParseImage( node );
```

```
        if ( node.Name == "item" )
            rssChannel.Items.Add( ParseItem( node ) );
    }

    return rssChannel;
}

private ChannelImage ParseImage( XmlNode node )
{
    ChannelImage channelImage = new ChannelImage();

    foreach ( XmlNode imageNode in node.ChildNodes )
    {
        //TODO: parse do atributo rdf:about - realmente necessário?

        if ( imageNode.Name == "title" )
            channelImage.Title = imageNode.InnerText;

        if ( imageNode.Name == "link" )
            channelImage.Link = imageNode.InnerText;

        if ( imageNode.Name == "url" )
            channelImage.Url = imageNode.InnerText;
    }

    return channelImage;
}

private RssItem ParseItem( XmlNode node )
{
    RssItem rssItem = new RssItem();

    foreach ( XmlNode itemNode in node.ChildNodes )
    {
        //TODO: parse do atributo rdf:about - realmente necessário?

        if ( itemNode.Name == "dc:identifier" )
            rssItem.Guid = itemNode.InnerText;

        if ( itemNode.Name == "title" )
            rssItem.Title = itemNode.InnerText;

        if ( itemNode.Name == "link" )
            rssItem.Link = itemNode.InnerText;

        if ( itemNode.Name == "description" )
            rssItem.Description = itemNode.InnerText;

        if ( itemNode.Name == "dc:creator" )
            rssItem.Author = itemNode.InnerText;

        if ( itemNode.Name == "dc:date" )
            rssItem.PubDate = ParseDate( itemNode );

        if ( itemNode.Name == "dc:subject" )
            rssItem.Categories.Add( ParseCategory( itemNode ) );

        if ( itemNode.Name == "dc:source" )
            rssItem.Source = ParseSource( itemNode );
    }

    return rssItem;
}
```

```
        private CultureInfo ParseLanguage( XmlNode node )
        {
            try
            {
                return System.Globalization.CultureInfo.GetCultureInfo( node.InnerText );
            }
            catch
            {
                /*gravar log.*/
            }

            return null;  //é executado somente quando ocorre um erro ao obter a informação de
localização
        }

        private Nullable<DateTime> ParseDate( XmlNode node )
        {
            try
            {
                return DateTime.Parse( node.InnerText );
            }
            catch
            {
                /*gravar log*/
            }

            return null;
        }

        private RssCategory ParseCategory( XmlNode node )
        {
            RssCategory rssCategory = new RssCategory();
            rssCategory.Value = node.InnerText;
            return rssCategory;
        }

        private ItemSource ParseSource( XmlNode node )
        {
            ItemSource itemSource = new ItemSource();
            itemSource.Value = node.InnerText;
            return itemSource;
        }

        #endregion
    }
}
```

Atom10Parser.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Xml;
using System.Collections.Generic;
using System.Globalization;
using Aurora.Feed.Types.Atom;


namespace Aurora.Feed.Parsers.Atom
{
    public class Atom10Parser
```

```
    {
        #region Constructor

        public Atom10Parser()
        {
            //construtor da classe Atom10Parser.
        }

        #endregion

        #region Parser

        public AtomFeed Parse( XmlDocument xmlDocument )
        {
            AtomFeed atomFeed = new AtomFeed();

            atomFeed.FeedType = FeedType.Atom10;

            if ( xmlDocument.FirstChild.NodeType == XmlNodeType.XmlDeclaration )
            {
                XmlDeclaration xmlDeclaration = ( XmlDeclaration )xmlDocument.FirstChild;
                atomFeed.XmlVersion = xmlDeclaration.Version;
                atomFeed.XmlEncoding = xmlDeclaration.Encoding;
            }

            XmlElement root = xmlDocument.DocumentElement;

            if ( root.Attributes.Count > 0 )
            {
                XmlAttribute feedLanguage = root.Attributes[ "xml:lang" ];
                if ( null != feedLanguage )
                    atomFeed.Language = System.Globalization.CultureInfo.GetCultureInfo(
feedLanguage.Value );
            }

            foreach ( XmlNode node in root.ChildNodes )
            {
                if ( node.Name == "id" )
                    atomFeed.Id = node.InnerText;

                if ( node.Name == "title" )
                    atomFeed.Title = ParseTextConstruct( node );

                if ( node.Name == "updated" )
                    atomFeed.Updated = ParseDate( node );

                if ( node.Name == "author" )
                    atomFeed.Authors.Add( ParsePersonConstruct( node ) );

                if ( node.Name == "link" )
                    atomFeed.Links.Add( ParseLink( node ) );

                if ( node.Name == "category" )
                    atomFeed.Categories.Add( ParseCategory( node ) );

                if ( node.Name == "contributor" )
                    atomFeed.Contributors.Add( ParsePersonConstruct( node ) );

                if ( node.Name == "generator" )
                    atomFeed.Generator = ParseGenerator( node );

                if ( node.Name == "icon" )
                    atomFeed.Icon = node.InnerText;
```

```
        if ( node.Name == "logo" )
            atomFeed.Logo = node.InnerText;

        if ( node.Name == "rights" )
            atomFeed.Rights = ParseTextConstruct( node );

        if ( node.Name == "subtitle" )
            atomFeed.Subtitle = ParseTextConstruct( node );

        if ( node.Name == "entry" )
            atomFeed.Entries.Add( ParseEntry( node ) );
    }

    return atomFeed;
}

private AtomTextConstruct ParseTextConstruct( XmlNode node )
{
    AtomTextConstruct textConstruct = new AtomTextConstruct();

    if ( node.Attributes.Count > 0 )
    {
        XmlAttribute attribute = node.Attributes[ "type" ];
        if ( null != attribute )
            textConstruct.Type = attribute.Value;
    }

    textConstruct.Value = node.InnerXml;

    return textConstruct;
}

private AtomPersonConstruct ParsePersonConstruct( XmlNode node )
{
    AtomPersonConstruct personConstruct = new AtomPersonConstruct();

    foreach ( XmlNode constructNode in node.ChildNodes )
    {
        if ( constructNode.Name == "name" )
            personConstruct.Name = constructNode.InnerText;

        if ( constructNode.Name == "uri" )
            personConstruct.Uri = constructNode.InnerText;

        if ( constructNode.Name == "email" )
            personConstruct.Email = constructNode.InnerText;
    }

    return personConstruct;
}

private Nullable<DateTime> ParseDate( XmlNode node )
{
    try
    {
        return DateTime.Parse( node.InnerText );
    }
    catch
    {
        /*gravar log*/
    }
```

```
            return null;
        }

        private FeedLink ParseLink( XmlNode node )
        {
            FeedLink feedLink = new FeedLink();

            if ( node.Attributes.Count > 0 )
            {
                XmlAttribute attribute = node.Attributes[ "href" ];
                if ( null != attribute )
                    feedLink.Href = attribute.Value;

                attribute = node.Attributes[ "rel" ];
                if ( null != attribute )
                    feedLink.Rel = attribute.Value;

                attribute = node.Attributes[ "type" ];
                if ( null != attribute )
                    feedLink.Type = attribute.Value;

                attribute = node.Attributes[ "hreflang" ];
                if ( null != attribute )
                {
                    try
                    {
                        feedLink.Hreflang = System.Globalization.CultureInfo.GetCultureInfo(
attribute.Value );
                    }
                    catch
                    {
                        /*gravar log.*/
                    }
                }

                attribute = node.Attributes[ "title" ];
                if ( null != attribute )
                    feedLink.Title = attribute.Value;

                attribute = node.Attributes[ "length" ];
                if ( null != attribute )
                {
                    try
                    {
                        feedLink.Length = long.Parse( attribute.Value );
                    }
                    catch
                    {
                        /*gravar log*/
                    }
                }
            }

            return feedLink;
        }

        private FeedCategory ParseCategory( XmlNode node )
        {
            FeedCategory feedCategory = new FeedCategory();

            if ( node.Attributes.Count > 0 )
            {
                XmlAttribute attribute = node.Attributes[ "term" ];
```

```
        if ( null != attribute )
            feedCategory.Term = attribute.Value;

        attribute = node.Attributes[ "scheme" ];
        if ( null != attribute )
            feedCategory.Scheme = attribute.Value;

        attribute = node.Attributes[ "label" ];
        if ( null != attribute )
            feedCategory.Label = attribute.Value;
    }

    return feedCategory;
}

private FeedGenerator ParseGenerator( XmlNode node )
{
    FeedGenerator feedGenerator = new FeedGenerator();

    if ( node.Attributes.Count > 0 )
    {
        XmlAttribute attribute = node.Attributes[ "uri" ];
        if ( null != attribute )
            feedGenerator.Uri = attribute.Value;

        attribute = node.Attributes[ "version" ];
        if ( null != attribute )
            feedGenerator.Version = attribute.Value;
    }

    feedGenerator.Value = node.InnerText;

    return feedGenerator;
}

private AtomEntry ParseEntry( XmlNode node )
{
    AtomEntry atomEntry = new AtomEntry();

    foreach ( XmlNode entryNode in node.ChildNodes )
    {
        if ( entryNode.Name == "id" )
            atomEntry.Id = entryNode.InnerText;

        if ( entryNode.Name == "title" )
            atomEntry.Title = ParseTextConstruct( entryNode );

        if ( entryNode.Name == "updated" )
            atomEntry.Updated = ParseDate( entryNode );

        if ( entryNode.Name == "author" )
            atomEntry.Authors.Add( ParsePersonConstruct( entryNode ) );

        if ( entryNode.Name == "content" )
            atomEntry.Content = ParseContent( entryNode );

        if ( entryNode.Name == "summary" )
            atomEntry.Summary = ParseTextConstruct( entryNode );

        if ( entryNode.Name == "link" )
            atomEntry.Links.Add( ParseLink( entryNode ) );

        if ( entryNode.Name == "category" )
```

```
            atomEntry.Categories.Add( ParseCategory( entryNode ) );

        if ( entryNode.Name == "contributor" )
            atomEntry.Contributors.Add( ParsePersonConstruct( entryNode ) );

        if ( entryNode.Name == "published" )
            atomEntry.Published = ParseDate( entryNode );

        if ( entryNode.Name == "source" )
            atomEntry.Source = ParseSource( entryNode );

        if ( entryNode.Name == "rights" )
            atomEntry.Rights = ParseTextConstruct( entryNode );
    }

    return atomEntry;
}

private EntryContent ParseContent( XmlNode node )
{
    EntryContent entryContent = new EntryContent();

    if ( node.Attributes.Count > 0 )
    {
        XmlAttribute attribute = node.Attributes[ "type" ];
        if ( null != attribute )
            entryContent.Type = node.Value;

        attribute = node.Attributes[ "src" ];
        if ( null != attribute )
            entryContent.Source = node.Value;
    }

    entryContent.Value = node.InnerXml;

    return entryContent;
}

private EntrySource ParseSource( XmlNode node )
{
    EntrySource entrySource = new EntrySource();

    foreach ( XmlNode sourceNode in node.ChildNodes )
    {
        if ( sourceNode.Name == "id" )
            entrySource.Id = sourceNode.InnerText;

        if ( sourceNode.Name == "title" )
            entrySource.Title = ParseTextConstruct( sourceNode );

        if ( sourceNode.Name == "updated" )
            entrySource.Updated = ParseDate( sourceNode );

        if ( sourceNode.Name == "author" )
            entrySource.Authors.Add( ParsePersonConstruct( sourceNode ) );

        if ( sourceNode.Name == "link" )
            entrySource.Links.Add( ParseLink( sourceNode ) );

        if ( sourceNode.Name == "category" )
            entrySource.Categories.Add( ParseCategory( sourceNode ) );

        if ( sourceNode.Name == "contributor" )
```

```
                    entrySource.Contributors.Add( ParsePersonConstruct( sourceNode ) );

                if ( node.Name == "generator" )
                    entrySource.Generator = ParseGenerator( sourceNode );

                if ( node.Name == "icon" )
                    entrySource.Icon = sourceNode.InnerText;

                if ( node.Name == "logo" )
                    entrySource.Logo = sourceNode.InnerText;

                if ( sourceNode.Name == "rights" )
                    entrySource.Rights = ParseTextConstruct( sourceNode );

                if ( node.Name == "subtitle" )
                    entrySource.Subtitle = ParseTextConstruct( sourceNode );
            }

            return entrySource;
        }

        #endregion
    }
}
```

FeedConverter.cs

```
using System;
using System.Data;
using System.Configuration;
using Aurora.Feed.Types;
using Aurora.Feed.Types.Rss;
using Aurora.Feed.Types.Atom;
using Aurora.Feed.Converters.Rss;
using Aurora.Feed.Converters.Atom;

public class FeedConverter
{
    #region Constructor

    public FeedConverter()
    {
               //construtor da classe FeedConverter.
    }

    #endregion


    #region Converter

    public static NewsFeed Convert( HardwiredFeed hardwiredFeed )
    {
        if ( hardwiredFeed.FeedType == FeedType.Atom10 )
        {
            AtomConverter feedConverter = new AtomConverter();
            return feedConverter.Convert( ( AtomFeed ) hardwiredFeed );
        }

        if ( hardwiredFeed.FeedType == FeedType.RSS20 )
        {
            RssConverter feedConverter = new RssConverter();
            return feedConverter.Convert( ( RssChannel ) hardwiredFeed );
```

```
        }

        if ( hardwiredFeed.FeedType == FeedType.RSS091 )
        {
            RssConverter feedConverter = new RssConverter();
            return feedConverter.Convert( ( RssChannel ) hardwiredFeed );
        }

        if ( hardwiredFeed.FeedType == FeedType.RSS10 )
        {
            RssConverter feedConverter = new RssConverter();
            return feedConverter.Convert( ( RssChannel ) hardwiredFeed );
        }

        if ( hardwiredFeed.FeedType == FeedType.RSS092 )
        {
            RssConverter feedConverter = new RssConverter();
            return feedConverter.Convert( ( RssChannel ) hardwiredFeed );
        }

        return null; //não foi possível identificar o tipo do feed passado como HardwiredFeed
    }

    #endregion
}
```

RssConverter.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Collections.Generic;
using System.Globalization;
using Aurora.Feed.Types;
using Aurora.Feed.Types.Rss;


namespace Aurora.Feed.Converters.Rss
{
    public class RssConverter
    {
        #region Constructor

        public RssConverter()
        {
            // TODO: Add constructor logic here
        }

        #endregion

        #region Converter

        public NewsFeed Convert( RssChannel rssChannel )
        {
            NewsFeed newsFeed = new NewsFeed();

            newsFeed.FeedUrl = rssChannel.FeedUrl;
            newsFeed.Title = rssChannel.Title;
            newsFeed.Description = rssChannel.Description;
            newsFeed.Link = ConvertLink( rssChannel.Link );
            newsFeed.Language = rssChannel.Language;
            newsFeed.Image = rssChannel.Image.Url;
```

```
    newsFeed.Copyright = rssChannel.Copyright;
    newsFeed.Generator = rssChannel.Generator;
    newsFeed.Updated = rssChannel.LastBuildDate;
    newsFeed.Author = rssChannel.ManagingEditor;
    newsFeed.Categories = ConvertCategories( rssChannel.Categories );
    newsFeed.Items = ConvertItems( rssChannel.Items, newsFeed.FeedUrl );

    return newsFeed;
}

private List<NewsCategory> ConvertCategories( List<RssCategory> rssCategories )
{
    if ( null != rssCategories )
    {
        List<NewsCategory> newsCategories = new List<NewsCategory>();

        foreach ( RssCategory rssCategory in rssCategories )
        {
            NewsCategory newsCategory = new NewsCategory();

            newsCategory.Term = rssCategory.Value;
            newsCategory.Domain = rssCategory.Domain;

            newsCategories.Add( newsCategory );
        }

        return newsCategories;
    }
    else
        return null;
}

private List<NewsItem> ConvertItems( List<RssItem> rssItems, string feedUrl )
{
    if ( null != rssItems )
    {
        List<NewsItem> newsItems = new List<NewsItem>();

        foreach ( RssItem rssItem in rssItems )
        {
            NewsItem newsItem = new NewsItem();

            newsItem.ItemUrl = rssItem.Link;
            newsItem.FeedUrl = feedUrl;
            newsItem.Title = rssItem.Title;
            newsItem.Content = rssItem.Description;
            newsItem.Link = ConvertLink( rssItem.Link );
            newsItem.Author = rssItem.Author;
            newsItem.Published = rssItem.PubDate;
            newsItem.Categories = ConvertCategories( rssItem.Categories );
            newsItem.Source = ConvertSource( rssItem.Source );
            newsItem.Enclosure = ConvertEnclosure( rssItem.Enclosure );

            newsItems.Add( newsItem );
        }

        return newsItems;
    }
    else
        return null;
}

private NewsLink ConvertLink( string link )
```

```
        {
            if ( null != link )
            {
                NewsLink newsLink = new NewsLink();
                newsLink.Url = link;
                return newsLink;
            }
            else
                return null;
        }

        private NewsLink ConvertSource( ItemSource itemSource )
        {
            if ( null != itemSource )
            {
                NewsLink newsLink = new NewsLink();
                newsLink.Url = itemSource.Url;
                newsLink.Title = itemSource.Value;
                return newsLink;
            }
            else
                return null;
        }

        private NewsLink ConvertEnclosure( ItemEnclosure itemEnclosure )
        {
            if ( null != itemEnclosure )
            {
                NewsLink newsLink = new NewsLink();
                newsLink.Url = itemEnclosure.Url;
                newsLink.Type = itemEnclosure.Type;
                newsLink.Length = itemEnclosure.Length;
                return newsLink;
            }
            else
                return null;
        }

        #endregion
    }
}
```

AtomConverter.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Collections.Generic;
using System.Globalization;
using Aurora.Feed.Types;
using Aurora.Feed.Types.Atom;


namespace Aurora.Feed.Converters.Atom
{
    public class AtomConverter
    {
        #region Constructor

        public AtomConverter()
        {
            // TODO: Add constructor logic here
```

```
        }

        #endregion

        #region Converter

        public NewsFeed Convert( AtomFeed atomFeed )
        {
            NewsFeed newsFeed = new NewsFeed();

            newsFeed.FeedUrl = atomFeed.Id;
            newsFeed.Title = ConvertTextConstruct( atomFeed.Title );
            newsFeed.Description = ConvertTextConstruct( atomFeed.Subtitle );
            newsFeed.Link = ConvertLink( atomFeed.Links );
            newsFeed.Language = atomFeed.Language;
            newsFeed.Image = atomFeed.Logo;
            newsFeed.Copyright = ConvertTextConstruct( atomFeed.Rights );
            newsFeed.Generator = ConvertGenerator( atomFeed.Generator );
            newsFeed.Updated = atomFeed.Updated;
            newsFeed.Author = ConvertAuthor( atomFeed.Authors );
            newsFeed.Categories = ConvertCategories( atomFeed.Categories );
            newsFeed.Items = ConvertEntries( atomFeed.Entries, newsFeed.FeedUrl );

            return newsFeed;
        }

        private string ConvertTextConstruct( AtomTextConstruct textConstruct )
        {
            if ( null != textConstruct )
                return textConstruct.Value;
            else
                return null;
        }

        private string ConvertGenerator( FeedGenerator feedGenerator )
        {
            if ( null != feedGenerator )
                return feedGenerator.Value;
            else
                return null;
        }

        private NewsLink ConvertLink( List<FeedLink> links )
        {
            if ( null != links )
            {
                foreach ( FeedLink link in links )
                {
                    if ( link.Rel.ToLower() == "alternate" )
                    {
                        NewsLink newsLink = new NewsLink();

                        newsLink.Url = link.Href;
                        newsLink.Title = link.Title;

                        return newsLink;
                    }
                }
                return null; //não encontrou um link com relationship "alternate" entre os
links da lista
            }
            else
                return null;
```

```
}

private string ConvertAuthor( List<AtomPersonConstruct> authors )
{
    if ( null != authors )
    {
        return authors[ 0 ].Email;
    }
    else
        return null;
}

private List<NewsCategory> ConvertCategories( List<FeedCategory> atomCategories )
{
    if ( null != atomCategories )
    {
        List<NewsCategory> newsCategories = new List<NewsCategory>();

        foreach ( FeedCategory atomCategory in atomCategories )
        {
            NewsCategory newsCategory = new NewsCategory();

            newsCategory.Term = atomCategory.Term;
            newsCategory.Domain = atomCategory.Scheme;

            newsCategories.Add( newsCategory );
        }

        return newsCategories;
    }
    else
        return null;
}

private List<NewsItem> ConvertEntries( List<AtomEntry> atomEntries, string feedUrl )
{
    if ( null != atomEntries )
    {
        List<NewsItem> newsItems = new List<NewsItem>();

        foreach ( AtomEntry atomEntry in atomEntries )
        {
            NewsItem newsItem = new NewsItem();

            newsItem.ItemUrl = atomEntry.Id;
            newsItem.FeedUrl = feedUrl;
            newsItem.Title = ConvertTextConstruct( atomEntry.Title );

            if ( null != atomEntry.Content )
                newsItem.Content = atomEntry.Content.Value;
            else
                if ( null != atomEntry.Summary )
                    newsItem.Content = atomEntry.Summary.Value;

            newsItem.Link = ConvertLink( atomEntry.Links );
            newsItem.Author = ConvertAuthor( atomEntry.Authors );
            newsItem.Published = atomEntry.Published;
            newsItem.Categories = ConvertCategories( atomEntry.Categories );
            newsItem.Source = ConvertSource( atomEntry.Links );
            newsItem.Enclosure = ConvertEnclosure( atomEntry.Links );

            newsItems.Add( newsItem );
        }
```

```
                return newsItems;
            }
            else
                return null;
        }

        private NewsLink ConvertSource( List<FeedLink> entryLinks )
        {
            if ( null != entryLinks )
            {
                foreach ( FeedLink link in entryLinks )
                {
                    if ( link.Rel.ToLower() == "via" )
                    {
                        NewsLink newsLink = new NewsLink();

                        newsLink.Url = link.Href;
                        newsLink.Title = link.Title;

                        return newsLink;
                    }
                }
                return null; //não encontrou um link com relationship "via" entre os links da
lista
            }
            else
                return null;
        }

        private NewsLink ConvertEnclosure( List<FeedLink> entryLinks )
        {
            if ( null != entryLinks )
            {
                foreach ( FeedLink link in entryLinks )
                {
                    if ( link.Rel.ToLower() == "enclosure" )
                    {
                        NewsLink newsLink = new NewsLink();

                        newsLink.Url = link.Href;
                        newsLink.Title = link.Title;
                        newsLink.Type = link.Type;
                        newsLink.Length = link.Length;

                        return newsLink;
                    }
                }
                return null; //não encontrou um link com relationship "enclosure" entre os
links da lista
            }
            else
                return null;
        }

        #endregion
    }
}
```

KeywordExtractor.cs

```
using System;
```

```
using System.Data;
using System.Configuration;
using System.Text;
using System.Text.RegularExpressions;
using System.Web;
using Aurora.Feed.Types;
using System.Collections.Generic;

namespace Aurora.Feed.Extractor
{
    public class KeywordExtractor
    {
        private static string[] stoplist = new string[] { "abaixo", "acima", "acolá",
“ademais", "adiante", "afinal", "afora", "agora", "ainda", "algo", "algum", "alguma",
"algumas", "alguns", "alguém", "ali", "aliás", "além", "amanhã", “ano", "ante", "anteontem",
"antes", “ao", "aos", "apenas", "apesar", "após", "aquele", "aqueles", "aqui", "aquilo",
"aquém", "as", "assaz", "assim", "através", "atrás", "até", "avos", "aí", "bastante", "bem",
"bilhão", "bilionésimo", "bimestre", "boa", "breve", "cada", "catorze", "causa", "cem",
"centena", “centésimo", “certa", “certamente", "certas", "certeza", "certo", "certos", "cima",
“cinco", "cinqüenta", “com", “comigo", "como", "completamente", “conforme", "conosco",
“conquanto", “consigo", "contigo", "contra", "contudo", "convosco", "cuja", "cujas", “cujo",
“cujos", “cá", “da", “daquela", “daquelas", “daquele", "daqueles", “daquilo", “darão", “das",
“de", “decerto", “defronte", “deixa", "deixar", "deixem", "dela", "delas", "dele", "deles",
"demais", "demasia", “demasiadamente", “demasiado", “dentro", “depois", "depressa", "der",
"desde", "despeito", "dessa", "dessas", "desse", "desses", "desta", "destas", "deste",
“destes", “detrás", “devagar", “deve", "deveras", “devido", “dez", “dezena", “dezenove",
“dezesseis", “dezessete", “dezoito", "diante", "digo", "disse", "disso", "disto", "diz",
"dizem", "dizer", "do", "dobro", “dois", "dos", "doze", "duplo", "durante", "duzentos", "dão",
"décimo", "eis", "ela", "elas", "ele", "eles", "em", "embaixo", "embora", "enquanto",
"entanto", “entre", “entremeio", "entretanto", “então", “era", “eram", "essa", "essas",
"esse", "esses", "esta", "estar", "estará", "estas", "estava", "este", “estes", “esteve",
“estiveram", “estiverem", “está", “estão", “etc", “eu", “excessivamente", "excesso", "exceto",
"exceção", "fará", "farão", "faz", "fazem", "fazer", "felizmente", "fica", "ficaram", "ficou",
“fim", “foi", "for", “fora", “foram", “forem", “forma", “frente", “graças", “haviam", “hoje",
“houveram", “há", "inclusive", "incontestavelmente", "infelizmente", "ir", “isso", “isto",
“jamais", "jeito", “junto", “já", "lentamente", "lhe", "lhes", "logo", "longe", "lá", "mais",
"mas", "me", "mediante", "melhor", “menos", “mesma", “mesmo", “meu", “meus", "mil", “milhão",
“milionésimo", “milésimo", “mim", “minha", “minhas", “muita", "muitas", "muito", "muitos",
"na", "nada", "naquela", “naquelas", “naquele", “naqueles", “naquilo", “nas", “nela", “nelas",
“nele", "neles", "nem", "nenhum", "nenhuma", "nenhumas", "nenhuns", "nessa", "nessas",
"nesse", "nesses", "nesta", "nestas", "neste", “nestes", "ninguém", "nisso", "nisto", "no",
"nono", "nos", "nossa", "nossas", "nosso", "nossos", "nove", "novecentos", "noventa", "novo",
"num", "numa", "nunca", "não", "nós", "oitavo", "oitenta", "oito", "oitocentos", "onde",
"ontem", "onze", "ora", "os", “ou", “outra", “outras", “outrem", “outro", "outrora", "outros",
"outrossim", "par", "para", "pela", "pelas", "pelo", "pelos", "per", "perante", "perto",
"pode", “podem", "poderia", "poderá", "pois", "por", "porquanto", "porque", "porquê",
"portanto", "porventura", "porém", "pouca", "poucas", "pouco", "poucos", "pressas",
“primeiro", “propósito", "provavelmente", “próprio", “próprios", “quais", “quaisquer", “qual",
“qualquer", “quando", “quanta", "quantas", "quanto", "quantos", "quarenta", "quarto", "quase",
"quatro", "quatrocentos", "que", "queira", "quem", "quer", "querem", "querer", "quinhentos",
“quinto", “quinze", “quádruplo", “quê", “quíntuplo", “raramente", “realmente", "repente",
"respeito", "restante", "resto", "saber", "salvante", "salvo", "se", "segundo", “seis",
“seiscentos", “seja", “sem", "semestre", “sempre", “senhor", “senhora", “senão", “sequer",
“ser", "seria", "será", "serão", "sessenta", "sete", "setecentos", "setenta", "seu", "seus",
"sexto", “si", “sido", “sim", “sob", "sobre", "sobretudo", "somente", "sua", "suas",
"sucessivamente", "são", "sétimo", "só", "súbito", "tal", "talvez", “também", “tampouco",
“tanta", “tantas", "tanto", “tantos", “tarde", "te", "tem", "tempos", "ter", "terceiro",
"terá", "terão", "terço", "teu", "teus", "teve", "ti", "toa", "toda", "todas", "todavia",
“todo", “todos", “treze", "trezentos", "trinca", "trinta", "triplo", "trás", "três", "tu",
"tua", "tuas", "tudo", "tão", "têm", "um", "uma", "umas", "uns", “usam", “vai", “veja", “vem",
“vez", "vezes", "vinte", "visto", "você", "vocês", "vos", "vossa", "vossas", "vossos",
"várias", "vários", "vão", vêm", "vós", "àquela", "àquelas", "àquele", "àqueles", "àquilo",
"às", "ínterim" };
```

```
        public KeywordExtractor()
        {
            //construtor
        }

        public static NewsFeed ExtractKeywords( NewsFeed newsFeed )
        {
            List<string> keywords;

            foreach ( NewsItem newsItem in newsFeed.Items )
            {
                keywords =  new List<string>();
                keywords.AddRange( ExtractKeywords( newsItem.Title ) );
                keywords.AddRange( ExtractKeywords( newsItem.Content ) );

                foreach ( string keyword in keywords )
                {
                    newsItem.AddKeyword( keyword );
                }
            }

            return newsFeed;
        }

        private static List<string> ExtractKeywords( string content )
        {
            List<string> keywords;
            content = NormalizeContent( content );
            keywords = RemoveStopWords( content );
            return keywords;
        }

        private static string NormalizeContent( string content )
        {
            content = HttpUtility.HtmlDecode( content );

            string pattern = @"<script(.|\n)*?</script>"; //remove as tags de script
JavaScript
            content = Regex.Replace( content, pattern, " " );

            pattern = @"<style(.|\n)*?</style>"; //remove as tags de estilo CSS
            content = Regex.Replace( content, pattern, " " );

            pattern = @"<!--(.|\n)*?-->";
            content = Regex.Replace( content, pattern, " " );

            pattern = @"<img(.|\n)*?>";
            content = Regex.Replace( content, pattern, " " );

            pattern = @"<font(.|\n)*?>|</font>";
            content = Regex.Replace( content, pattern, " " );

            pattern = @"<a(.|\n)*?>|</a>";
            content = Regex.Replace( content, pattern, " " );

            pattern = @"<td(.|\n)*?>|</td>"; //remove as tags de TD
            content = Regex.Replace( content, pattern, " " );

            pattern = @"<tr(.|\n)*?>|</tr>";
            content = Regex.Replace( content, pattern, " " );

            pattern = @"<table(.|\n)*?>|</table>";
```

```
    content = Regex.Replace( content, pattern, " " );

    pattern = @"<div(.|\n)*?>|</div>";
    content = Regex.Replace( content, pattern, " " );

    pattern = @"<select(.|\n)*?</select>";
    content = Regex.Replace( content, pattern, " " );

    pattern = @"<br\s*/?>";
    content = Regex.Replace( content, pattern, " " );

    pattern = @"<input(.|\n)*?>";
    content = Regex.Replace( content, pattern, " " );

    pattern = @"<link(.|\n)*?>";
    content = Regex.Replace( content, pattern, " " );

    pattern = @"<meta(.|\n)*?>";
    content = Regex.Replace( content, pattern, " " );

    pattern = @"<(.|\n)*?>";
    content = Regex.Replace( content, pattern, " " );

    pattern = @"[-]{2,}";
    content = Regex.Replace( content, pattern, " " );

    pattern = @"-\s";
    content = Regex.Replace( content, pattern, " " );

    pattern = @"\s-";
    content = Regex.Replace( content, pattern, " " );

    pattern = @"[^0-9A-Za-zÁÀÂÃÉÊÍÓÔÕÚÜÇáàâãéêíóôõúüç-]";
    content = Regex.Replace( content, pattern, " " );

    pattern = @"\d{1,2}h\d{1,2}";
    content = Regex.Replace( content, pattern, " " );

    pattern = @"\s[\d-?]+";
    content = Regex.Replace( content, pattern, " " );

    pattern = @"\s.{1}\s";
    content = Regex.Replace( content, pattern, " " );

    pattern = @"[\s]{2,}";
    content = Regex.Replace( content, pattern, " " );

    content = content.ToLower();

    return content.Trim();
}

private static List<string> RemoveStopWords( string content )
{
    List<string> keywords = new List<string>( content.Split( ' ' ) );

    keywords.Remove( string.Empty );

    foreach ( string stopword in stoplist )
    {
        for ( int i = keywords.Count - 1; i > -1; i-- )
        {
            if ( keywords[ i ].Equals( stopword ) )
```

```
                    keywords.RemoveAt( i );
                }
            }


            return keywords;
        }
    }
}
```

User.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Aurora.Types
{
    public class User
    {
        #region Attributes

        private string id;
        private int newsItemsReviewed;
        private int interestingNewsItems;
        private int notInterestingNewsItems;
        private float interestingFrequency;
        private float notInterestingFrequency;

        #endregion

        #region Properties

        public string Id
        {
            get { return this.id; }
            set { this.id = value; }
        }

        public int NewsItemsReviewed
        {
            get { return this.newsItemsReviewed; }
            set { this.newsItemsReviewed = value; }
        }

        public int InterestingNewsItems
        {
            get { return this.interestingNewsItems; }
            set { this.interestingNewsItems = value; }
        }

        public int NotInterestingNewsItems
        {
            get { return this.notInterestingNewsItems; }
            set { this.notInterestingNewsItems = value; }
        }

        public float InterestingFrequency
        {
            get { return this.interestingFrequency; }
            set { this.interestingFrequency = value; }
```

```
        }

        public float NotInterestingFrequency
        {
            get { return this.notInterestingFrequency; }
            set { this.notInterestingFrequency = value; }
        }

        #endregion

        #region Constructor

        public User()
        {
            // TODO: Add constructor logic here
        }

        public User( string id )
        {
            this.id = id;
            this.newsItemsReviewed = 0;
            this.interestingNewsItems = 0;
            this.notInterestingNewsItems = 0;
        }

        public User( string id, int itemsReviewed, int interestingItems, int
notInterestingItems )
        {
            this.id = id;
            this.newsItemsReviewed = itemsReviewed;
            this.interestingNewsItems = interestingItems;
            this.notInterestingNewsItems = notInterestingItems;
        }

        #endregion
    }
}
```

UserController.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using Aurora.DAO;
using Aurora.Types;

namespace Aurora.Controller
{
    public class UserController
    {
        public static void add( User user )
        {
            UserDAO.save( user );
        }

        public static void remove( User user )
        {
            UserDAO.delete( user );
        }

        public static void remove( string userId )
        {
```

```
            UserDAO.delete( userId );
        }

        public static List<User> getUsers()
        {
            List<User> users = UserDAO.getUsers();

            foreach ( User user in users )
            {
                try { user.InterestingFrequency = user.InterestingNewsItems /
user.NewsItemsReviewed; }
                catch { user.InterestingFrequency = 0; }

                try { user.NotInterestingFrequency = user.NotInterestingNewsItems /
user.NewsItemsReviewed; }
                catch { user.NotInterestingFrequency = 0; }
            }

            return users;
        }

        public static User getUser( string userId )
        {
            User user = UserDAO.getUser( userId );

            try { user.InterestingFrequency = user.InterestingNewsItems /
user.NewsItemsReviewed; }
            catch { user.InterestingFrequency = 0; }

            try { user.NotInterestingFrequency = user.NotInterestingNewsItems /
user.NewsItemsReviewed; }
            catch { user.NotInterestingFrequency = 0; }

            return user;
        }

        public static void updateInterestingNewsItemsCount( string userId )
        {
            User user = UserDAO.getUser( userId );
            user.InterestingNewsItems += 1;
            user.NewsItemsReviewed += 1;
            UserDAO.update( user );
        }

        public static void updateNotInterestingNewsItemsCount( string userId )
        {
            User user = UserDAO.getUser( userId );
            user.NotInterestingNewsItems += 1;
            user.NewsItemsReviewed += 1;
            UserDAO.update( user );
        }
    }
}
```

NewsFeedController.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using Aurora.DAO;
using Aurora.Feed.Types;
using Aurora.Types;
```

```
namespace Aurora.Controller
{
    public class NewsFeedController
    {
        #region add

        /// <summary>
        /// Adiciona um novo feed (ou atualiza suas informações) no banco, sem atualizar as
notícias dele.
        /// </summary>
        /// <param name="newsFeed">NewsFeed com as informações a serem
inseridas/atualizadas</param>
        public static void add( NewsFeed newsFeed )
        {
            NewsFeedDAO.save( newsFeed );
        }

        /// <summary>
        /// Salva o conteúdo de um feed já existente, salvando também as notícias trazidas
dentro dele.
        /// </summary>
        /// <param name="newsFeed">NewsFeed com as informações a serem salvas</param>
        public static void save( NewsFeed newsFeed )
        {
            NewsFeedDAO.update( newsFeed );

            foreach ( NewsItem newsItem in newsFeed.Items )
                NewsItemController.add( newsItem );
        }

        public static void insertNewsFeedToUser( string userId, string feedUrl )
        {
            NewsFeedDAO.insertNewsFeedToUser( userId, feedUrl );
        }

        #endregion

        #region remove

        public static void remove( NewsFeed newsFeed )
        {
            NewsFeedDAO.delete( newsFeed );
        }

        public static void remove( string feedUrl )
        {
            NewsFeedDAO.delete( feedUrl );
        }

        public static void removeNewsFeedFromUser( string userId, string feedUrl )
        {
            NewsFeedDAO.deleteNewsFeedFromUser( userId, feedUrl );
        }

        #endregion

        #region get

        public static List<NewsFeed> getNewsFeeds()
        {
            return NewsFeedDAO.getNewsFeeds();
        }
```

```
        public static List<string> getNewsFeedsUrls()
        {
            return NewsFeedDAO.getNewsFeedsUrls();
        }

        public static List<NewsFeed> getNewsFeedsByUser( string userId )
        {
            return NewsFeedDAO.getNewsFeedsByUser( userId );
        }

        public static List<User> getUsersByNewsFeed( string feedUrl )
        {
            return NewsFeedDAO.getUsersByNewsFeed( feedUrl );
        }

        #endregion
    }
}
```

NewsItemController.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using Aurora.DAO;
using Aurora.Feed.Types;
using Aurora.Types;

namespace Aurora.Controller
{
    public class NewsItemController
    {
        #region add

        public static void add( NewsItem newsItem )
        {
            if ( !NewsItemDAO.exists( newsItem ) )
            {
                NewsItemDAO.insert( newsItem );

                foreach ( NewsItemKeyword keyword in newsItem.Keywords )
                    KeywordController.add( keyword );

                List<User> feedSubscribers = NewsFeedController.getUsersByNewsFeed(
newsItem.FeedUrl );

                foreach ( User user in feedSubscribers )
                {
                    NewsItemDAO.insertNewsItemToUser( newsItem, user );

                    foreach ( NewsItemKeyword keyword in newsItem.Keywords )
                        KeywordDAO.saveKeywordToUser( keyword, user.Id );
                }
            }
        }

        #endregion

        #region remove

        public static void remove( NewsItem newsItem )
```

```
        {
            NewsItemDAO.delete( newsItem );
        }

        public static void remove( string itemUrl, string feedUrl )
        {
            NewsItemDAO.delete( itemUrl, feedUrl );
        }

        #endregion

        #region get

        public static List<NewsItem> getNewsItems()
        {
            return NewsItemDAO.getNewsItems();
        }

        public static List<NewsItem> getNewsItems( string feedUrl )
        {
            return NewsItemDAO.getNewsItems( feedUrl );
        }

        public static NewsFeed getNewsItems( NewsFeed newsFeed )
        {
            return NewsItemDAO.getNewsItems( newsFeed );
        }

        public static List<NewsItem> getUnreadedNewsItems( string feedUrl, string userId )
        {
            List<NewsItem> newsItems = NewsItemDAO.getNewsItemsByUser( userId, feedUrl, false
);

            foreach ( NewsItem newsItem in newsItems )
                newsItem.Keywords = KeywordController.getKeywords( newsItem.FeedUrl,
newsItem.ItemUrl );

            return newsItems;
        }

        #endregion

        public static void markAsReaded( NewsItem newsItem, string userId )
        {
            NewsItemDAO.markAsReaded( newsItem, userId );
        }
    }
}
```

KeywordController.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using Aurora.DAO;
using Aurora.Feed.Types;
using Aurora.Types;

namespace Aurora.Controller
{
    public class KeywordController
    {
```

```
        public static void add( NewsItemKeyword keyword )
        {
            KeywordDAO.save( keyword ); //se a palavra chave não existir, insere-a no banco
            KeywordDAO.insertKeywordToNewsItem( keyword ); //insere a palavra-chave para a
notícia
        }

        public static List<NewsItemKeyword> getKeywords( string feedUrl, string itemUrl )
        {
            return KeywordDAO.getKeywordsByNewsItem( feedUrl, itemUrl );
        }

        public static void updateKeywordsAsReaded( List<NewsItemKeyword> keywords, string
userId )
        {
            foreach ( NewsItemKeyword keyword in keywords )
                KeywordDAO.updateAsReaded( keyword, userId );
        }

        public static void updateKeywordsAsNotReaded( List<NewsItemKeyword> keywords, string
userId )
        {
            foreach ( NewsItemKeyword keyword in keywords )
                KeywordDAO.updateAsNotReaded( keyword, userId );
        }

        public static List<UserKeyword> getUserKeywords( List<NewsItemKeyword> keywords,
string userId )
        {
            User user = UserController.getUser( userId );
            List<UserKeyword> userKeywords = KeywordDAO.getUserKeywords( keywords, userId );

            foreach ( UserKeyword userKeyword in userKeywords )
            {
                userKeyword.InterestingFrequency = userKeyword.InterestingOccurrences /
user.InterestingNewsItems;
                userKeyword.NotInterestingFrequency = userKeyword.NotInterestingOccurrences /
user.NotInterestingNewsItems;
            }

            return userKeywords;
        }
    }
}
```

UserDAO.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using ControlPanel.DAO;
using ControlPanel.DAO.DataSetAuroraTableAdapters;
using Aurora.Types;

namespace Aurora.DAO
{
    public class UserDAO
    {
        #region save

        public static void save( User user )
        {
```

```
            UsersTableAdapter usersAdapter = new UsersTableAdapter();
            DataSetAurora.UsersDataTable users = usersAdapter.GetUsers();

            DataSetAurora.UsersRow existingUser = users.FindByuserID( user.Id );

            if ( null == existingUser )
                insert( user );
            else
                update( user );
        }

        public static void insert( User user )
        {
            UsersTableAdapter usersAdapter = new UsersTableAdapter();
            try
            {
                usersAdapter.Insert( user.Id, 2, 1, 1 );
            }
            catch ( Exception e )
            {
                System.Windows.Forms.MessageBox.Show( "Erro na chamada de \"UserDAO.insert(
User user )\"\n\nDetalhes: " + e.Message, "Exceção interna na camada DAO",
System.Windows.Forms.MessageBoxButtons.OK, System.Windows.Forms.MessageBoxIcon.Error );
            }
        }

        public static void update( User user )
        {
            UsersTableAdapter usersAdapter = new UsersTableAdapter();
            try
            {
                usersAdapter.Update( user.Id, user.NewsItemsReviewed,
user.InterestingNewsItems, user.NotInterestingNewsItems );
            }
            catch ( Exception e )
            {
                System.Windows.Forms.MessageBox.Show( "Erro na chamada de \"UserDAO.update(
User user )\"\n\nDetalhes: " + e.Message, "Exceção interna na camada DAO",
System.Windows.Forms.MessageBoxButtons.OK, System.Windows.Forms.MessageBoxIcon.Error );
            }
        }

        #endregion

        #region delete

        public static void delete( User user )
        {
            UsersTableAdapter usersAdapter = new UsersTableAdapter();
            try
            {
                usersAdapter.Delete( user.Id );
            }
            catch ( Exception e )
            {
                System.Windows.Forms.MessageBox.Show( "Erro na chamada de \"UserDAO.delete(
User user )\"\n\nDetalhes: " + e.Message, "Exceção interna na camada DAO",
System.Windows.Forms.MessageBoxButtons.OK, System.Windows.Forms.MessageBoxIcon.Error );
            }
        }

        public static void delete( string userId )
        {
```

```
            UsersTableAdapter usersAdapter = new UsersTableAdapter();
            try
            {
                usersAdapter.Delete( userId );
            }
            catch ( Exception e )
            {
                System.Windows.Forms.MessageBox.Show( "Erro na chamada de \"UserDAO.delete(
string userId )\"\n\nDetalhes: " + e.Message, "Exceção interna na camada DAO",
System.Windows.Forms.MessageBoxButtons.OK, System.Windows.Forms.MessageBoxIcon.Error );
            }
        }

        #endregion

        #region get

        public static List<User> getUsers()
        {
            List<User> users = new List<User>();

            UsersTableAdapter usersAdapter = new UsersTableAdapter();
            DataSetAurora.UsersDataTable usersData = usersAdapter.GetUsers();

            foreach ( DataSetAurora.UsersRow userRow in usersData )
                users.Add( new User( userRow.userID, userRow.newsItemsReviewed,
userRow.interestingNewsItems, userRow.notInterestingNewsItems ) );

            return users;
        }

        public static User getUser( string userId )
        {
            UsersTableAdapter usersAdapter = new UsersTableAdapter();
            DataSetAurora.UsersDataTable usersData = usersAdapter.GetUser( userId );
            DataSetAurora.UsersRow userRow = usersData[ 0 ];

            return ( new User( userRow.userID, userRow.newsItemsReviewed,
userRow.interestingNewsItems, userRow.notInterestingNewsItems ) );
        }

        #endregion
    }
}
```

NewsFeedDAO.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Globalization;
using ControlPanel.DAO;
using ControlPanel.DAO.DataSetAuroraTableAdapters;
using Aurora.Feed.Types;
using Aurora.Types;

namespace Aurora.DAO
{
    public class NewsFeedDAO
    {
        #region save
```

```
        public static void save( NewsFeed newsFeed )
        {
            if ( !exists( newsFeed ) )
                insert( newsFeed );
            else
                update( newsFeed );
        }

        public static void insert( NewsFeed newsFeed )
        {
            NewsFeedsTableAdapter newsFeedsAdapter = new NewsFeedsTableAdapter();
            try
            {
                newsFeedsAdapter.Insert( newsFeed.FeedUrl, newsFeed.Title, newsFeed.Link.Url,
newsFeed.Description, newsFeed.Language.Name, newsFeed.Image, newsFeed.Copyright,
newsFeed.Generator, newsFeed.Updated, newsFeed.Author );
            }
            catch ( Exception e )
            {
                System.Windows.Forms.MessageBox.Show( "Erro na chamada de
\"NewsFeedDAO.insert( NewsFeed newsFeed )\"\n\nDetalhes: " + e.Message, "Exceção interna na
camada DAO", System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error );
            }
        }

        public static void update( NewsFeed newsFeed )
        {
            NewsFeedsTableAdapter newsFeedsAdapter = new NewsFeedsTableAdapter();
            try
            {
                newsFeedsAdapter.Update( newsFeed.FeedUrl, newsFeed.Title, newsFeed.Link.Url,
newsFeed.Description, newsFeed.Language.Name, newsFeed.Image, newsFeed.Copyright,
newsFeed.Generator, newsFeed.Updated, newsFeed.Author );
            }
            catch ( Exception e )
            {
                System.Windows.Forms.MessageBox.Show( "Erro na chamada de
\"NewsFeedDAO.update( NewsFeed newsFeed )\"\n\nDetalhes: " + e.Message, "Exceção interna na
camada DAO", System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error );
            }
        }

        public static void insertNewsFeedToUser( string userId, string feedUrl )
        {
            UsersNewsFeedsTableAdapter usersFeedsAdapter = new UsersNewsFeedsTableAdapter();
            try
            {
                usersFeedsAdapter.Insert( userId, feedUrl );
            }
            catch ( Exception e )
            {
                System.Windows.Forms.MessageBox.Show( "Erro na chamada de
\"NewsFeedDAO.insertNewsFeedToUser( string feedUrl, string userId )\"\n\nDetalhes: " +
e.Message, "Exceção interna na camada DAO", System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error );
            }
        }

        #endregion

        #region delete
```

```
        public static void delete( NewsFeed newsFeed )
        {
            NewsFeedsTableAdapter newsFeedsAdapter = new NewsFeedsTableAdapter();
            try
            {
                newsFeedsAdapter.Delete( newsFeed.FeedUrl );
            }
            catch ( Exception e )
            {
                System.Windows.Forms.MessageBox.Show( "Erro na chamada de
\"NewsFeedDAO.delete( NewsFeed newsFeed )\"\n\nDetalhes: " + e.Message, "Exceção interna na
camada DAO", System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error );
            }
        }

        public static void delete( string feedUrl )
        {
            NewsFeedsTableAdapter newsFeedsAdapter = new NewsFeedsTableAdapter();
            try
            {
                newsFeedsAdapter.Delete( feedUrl );
            }
            catch ( Exception e )
            {
                System.Windows.Forms.MessageBox.Show( "Erro na chamada de
\"NewsFeedDAO.delete( string feedUrl )\"\n\nDetalhes: " + e.Message, "Exceção interna na
camada DAO", System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error );
            }
        }

        public static void deleteNewsFeedFromUser( string userId, string feedUrl )
        {
            UsersNewsFeedsTableAdapter usersFeedsAdapter = new UsersNewsFeedsTableAdapter();
            try
            {
                usersFeedsAdapter.Delete( userId, feedUrl );
            }
            catch ( Exception e )
            {
                System.Windows.Forms.MessageBox.Show( "Erro na chamada de
\"NewsFeedDAO.deleteNewsFeedFromUser( string feedUrl, string userId )\"\n\nDetalhes: " +
e.Message, "Exceção interna na camada DAO", System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error );
            }
        }

        #endregion

        #region get

        public static List<NewsFeed> getNewsFeeds()
        {
            List<NewsFeed> newsFeeds = new List<NewsFeed>();

            NewsFeedsTableAdapter newsFeedsAdapter = new NewsFeedsTableAdapter();
            DataSetAurora.NewsFeedsDataTable newsFeedData = newsFeedsAdapter.GetNewsFeeds();

            foreach ( DataSetAurora.NewsFeedsRow newsFeedRow in newsFeedData )
            {
                NewsFeed newsFeed = new NewsFeed();
```

```
                newsFeed.FeedUrl = newsFeedRow.feedUrl;

                if ( !newsFeedRow.IstitleNull() )
                    newsFeed.Title = newsFeedRow.title;

                if ( !newsFeedRow.IslinkNull() )
                    newsFeed.Link = new NewsLink( newsFeedRow.link );

                if ( !newsFeedRow.IsdescriptionNull() )
                    newsFeed.Description = newsFeedRow.description;

                if ( !newsFeedRow.IslanguageNull() )
                    newsFeed.Language = CultureInfo.GetCultureInfo( newsFeedRow.language );

                if ( !newsFeedRow.IsimageUrlNull() )
                    newsFeed.Image = newsFeedRow.imageUrl;

                if ( !newsFeedRow.IscopyrightNull() )
                    newsFeed.Copyright = newsFeedRow.copyright;

                if ( !newsFeedRow.IsgeneratorNull() )
                    newsFeed.Generator = newsFeedRow.generator;

                if ( !newsFeedRow.IsupdatedNull() )
                    newsFeed.Updated = newsFeedRow.updated;

                if ( !newsFeedRow.IsauthorNull() )
                    newsFeed.Author = newsFeedRow.author;

                newsFeeds.Add( newsFeed );
            }

            return newsFeeds;
        }

        public static List<string> getNewsFeedsUrls()
        {
            List<string> newsFeedsUrls = new List<string>();

            NewsFeedsTableAdapter newsFeedsAdapter = new NewsFeedsTableAdapter();
            DataSetAurora.NewsFeedsDataTable newsFeedData = newsFeedsAdapter.GetNewsFeeds();

            foreach ( DataSetAurora.NewsFeedsRow newsFeedRow in newsFeedData )
                newsFeedsUrls.Add( newsFeedRow.feedUrl );

            return newsFeedsUrls;
        }

        public static List<NewsFeed> getNewsFeedsByUser( string userId )
        {
            List<NewsFeed> newsFeeds = new List<NewsFeed>();

            NewsFeedsTableAdapter newsFeedsAdapter = new NewsFeedsTableAdapter();
            DataSetAurora.NewsFeedsDataTable newsFeedData =
newsFeedsAdapter.GetNewsFeedsByUser( userId );

            foreach ( DataSetAurora.NewsFeedsRow newsFeedRow in newsFeedData )
            {
                NewsFeed newsFeed = new NewsFeed();
                newsFeed.FeedUrl = newsFeedRow.feedUrl;

                if ( !newsFeedRow.IstitleNull() )
                    newsFeed.Title = newsFeedRow.title;
```

```
                if ( !newsFeedRow.IslinkNull() )
                    newsFeed.Link = new NewsLink( newsFeedRow.link );

                if ( !newsFeedRow.IsdescriptionNull() )
                    newsFeed.Description = newsFeedRow.description;

                if ( !newsFeedRow.IslanguageNull() )
                    newsFeed.Language = CultureInfo.GetCultureInfo( newsFeedRow.language );

                if ( !newsFeedRow.IsimageUrlNull() )
                    newsFeed.Image = newsFeedRow.imageUrl;

                if ( !newsFeedRow.IscopyrightNull() )
                    newsFeed.Copyright = newsFeedRow.copyright;

                if ( !newsFeedRow.IsgeneratorNull() )
                    newsFeed.Generator = newsFeedRow.generator;

                if ( !newsFeedRow.IsupdatedNull() )
                    newsFeed.Updated = newsFeedRow.updated;

                if ( !newsFeedRow.IsauthorNull() )
                    newsFeed.Author = newsFeedRow.author;

                newsFeeds.Add( newsFeed );
            }

            return newsFeeds;
        }

        public static List<User> getUsersByNewsFeed( string feedUrl )
        {
            List<User> users = new List<User>();

            UsersNewsFeedsTableAdapter usersFeedsAdapter = new UsersNewsFeedsTableAdapter();
            DataSetAurora.UsersNewsFeedsDataTable usersFeedsData =
usersFeedsAdapter.GetUsersByNewsFeed( feedUrl );

            foreach ( DataSetAurora.UsersNewsFeedsRow usersFeedsRow in usersFeedsData )
            {
                User user = new User();
                user.Id = usersFeedsRow.userID;

                users.Add( user );
            }

            return users;
        }

        public static bool exists( NewsFeed newsFeed )
        {
            NewsFeedsTableAdapter newsFeedsAdapter = new NewsFeedsTableAdapter();
            DataSetAurora.NewsFeedsDataTable newsFeeds = newsFeedsAdapter.GetNewsFeeds();
            DataSetAurora.NewsFeedsRow feedExists = newsFeeds.FindByfeedUrl( newsFeed.FeedUrl
);

            if ( null != feedExists )
                return true;
            else
                return false;
        }
```

```
        #endregion
    }
}
```

NewsItemDAO.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Globalization;
using ControlPanel.DAO;
using ControlPanel.DAO.DataSetAuroraTableAdapters;
using Aurora.Feed.Types;
using Aurora.Types;

namespace Aurora.DAO
{
    public class NewsItemDAO
    {
        #region save

        public static void save( NewsItem newsItem )
        {
            if ( !exists( newsItem ) ) //se a notícia ainda não foi inserida no banco de dados
                insert( newsItem );
            else
                update( newsItem );
        }

        public static void insert( NewsItem newsItem )
        {
            NewsItemsTableAdapter newsItemsAdapter = new NewsItemsTableAdapter();
            try
            {
                newsItemsAdapter.Insert( newsItem.ItemUrl, newsItem.FeedUrl, newsItem.Title,
newsItem.Content, newsItem.Author, newsItem.Published );
            }
            catch ( Exception e )
            {
                System.Windows.Forms.MessageBox.Show( "Erro na chamada de
\"NewsItemDAO.insert( NewsItem newsItem )\"\n\nDetalhes: " + e.Message, "Exceção interna na
camada DAO", System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error );
            }
        }

        public static void insertNewsItemToUser( NewsItem newsItem, User user )
        {
            UsersNewsItemsTableAdapter usersItemsAdapter = new UsersNewsItemsTableAdapter();
            try
            {
                usersItemsAdapter.Insert( newsItem.ItemUrl, newsItem.FeedUrl, user.Id, false
);
            }
            catch ( Exception e )
            {
                System.Windows.Forms.MessageBox.Show( "Erro na chamada de
\"NewsItemDAO.insertNewsItemToUser( NewsItem newsItem, User user )\"\n\nDetalhes: " +
e.Message, "Exceção interna na camada DAO", System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error );
            }
        }
```

```
        #endregion

        #region get

        public static List<NewsItem> getNewsItemsByUser( string userId, string feedUrl, bool
readed )
        {
            List<NewsItem> newsItems = new List<NewsItem>();

            NewsItemsTableAdapter newsItemsAdapter = new NewsItemsTableAdapter();
            DataSetAurora.NewsItemsDataTable newsItemsData =
newsItemsAdapter.GetNewsItemsByUser( userId, feedUrl, readed );

            foreach ( DataSetAurora.NewsItemsRow newsItemRow in newsItemsData )
            {
                NewsItem newsItem = new NewsItem();

                newsItem.ItemUrl = newsItemRow.itemUrl;
                newsItem.FeedUrl = newsItemRow.feedUrl;

                if ( !newsItemRow.IstitleNull() )
                    newsItem.Title = newsItemRow.title;

                if ( !newsItemRow.IscontentNull() )
                    newsItem.Content = newsItemRow.content;

                if ( !newsItemRow.IsauthorNull() )
                    newsItem.Author = newsItemRow.author;

                if ( !newsItemRow.IspublishedNull() )
                    newsItem.Published = newsItemRow.published;

                newsItems.Add( newsItem );
            }

            return newsItems;
        }

        public static bool exists( NewsItem newsItem )
        {
            NewsItemsTableAdapter newsItemsAdapter = new NewsItemsTableAdapter();
            DataSetAurora.NewsItemsDataTable newsItems = newsItemsAdapter.GetNewsItem(
newsItem.ItemUrl, newsItem.FeedUrl );
            DataSetAurora.NewsItemsRow newsItemExists = newsItems.FindByitemUrlfeedUrl(
newsItem.ItemUrl, newsItem.FeedUrl );

            if ( null != newsItemExists ) //se a notícia já foi inserida no banco
                return true;
            else
                return false;
        }

        #endregion

        public static void markAsReaded( NewsItem newsItem, string userId )
        {
            UsersNewsItemsTableAdapter userNewsItemsAdapter = new
UsersNewsItemsTableAdapter();
            userNewsItemsAdapter.UpdateUserNewsItemAsReaded( newsItem.ItemUrl,
newsItem.FeedUrl, userId );
        }
    }
```

```
}
```

KeywordDAO.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using ControlPanel.DAO;
using ControlPanel.DAO.DataSetAuroraTableAdapters;
using Aurora.Feed.Types;
using Aurora.Types;

namespace Aurora.DAO
{
    public class KeywordDAO
    {
        #region save

        public static void save( NewsItemKeyword keyword )
        {
            if ( !exists( keyword ) )
                insert( keyword );
            else
                update( keyword );
        }

        public static void saveKeywordToUser( NewsItemKeyword keyword, string userId )
        {
            if ( !existsKeywordToUser( keyword, userId ) )
                insertKeywordToUser( keyword, userId );
        }

        public static void insert( NewsItemKeyword keyword )
        {
            KeywordsTableAdapter keywordsAdapter = new KeywordsTableAdapter();
            try
            {
                keywordsAdapter.Insert( keyword.Value );
            }
            catch ( Exception e )
            {
                System.Windows.Forms.MessageBox.Show( "Erro na chamada de \"KeywordDAO.insert(
NewsItemKeyword keyword )\"\n\nDetalhes: " + e.Message, "Exceção interna na camada DAO",
System.Windows.Forms.MessageBoxButtons.OK, System.Windows.Forms.MessageBoxIcon.Error );
            }
        }

        public static void insertKeywordToUser( NewsItemKeyword keyword, string userId )
        {
            DataSetAurora.KeywordsRow keywordRow = getKeywordByValue( keyword );
            UsersKeywordsTableAdapter usersKeywordsAdapter = new UsersKeywordsTableAdapter();
            try
            {
                usersKeywordsAdapter.Insert( userId, keywordRow.keywordID, 1, 1 );
            }
            catch ( Exception e )
            {
                System.Windows.Forms.MessageBox.Show( "Erro na chamada de \"KeywordDAO.insert(
NewsItemKeyword keyword )\"\n\nDetalhes: " + e.Message, "Exceção interna na camada DAO",
System.Windows.Forms.MessageBoxButtons.OK, System.Windows.Forms.MessageBoxIcon.Error );
            }
```

```
        }

        public static void insertKeywordToNewsItem( NewsItemKeyword keyword )
        {
            NewsItemsKeywordsTableAdapter itemsKeywordsAdapter = new
NewsItemsKeywordsTableAdapter();
            DataSetAurora.KeywordsRow keywordRow = getKeywordByValue( keyword );
            try
            {
                itemsKeywordsAdapter.Insert( keyword.ItemUrl, keyword.FeedUrl,
keywordRow.keywordID, keyword.Occurrences );
            }
            catch ( Exception e )
            {
                System.Windows.Forms.MessageBox.Show( "Erro na chamada de
\"KeywordDAO.insertKeywordToNewsItem( NewsItemKeyword keyword )\"\n\nDetalhes: " + e.Message,
"Exceção interna na camada DAO", System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error );
            }
        }

        public static void updateAsReaded( NewsItemKeyword keyword, string userId )
        {
            UsersKeywordsTableAdapter usersKeywordsAdapter = new UsersKeywordsTableAdapter();
            DataSetAurora.UsersKeywordsRow userKeywordRow = getUserKeyword( keyword, userId );
            userKeywordRow.interestingOccurrences += keyword.Occurrences;
            usersKeywordsAdapter.Update( userKeywordRow );
        }

        public static void updateAsNotReaded( NewsItemKeyword keyword, string userId )
        {
            UsersKeywordsTableAdapter usersKeywordsAdapter = new UsersKeywordsTableAdapter();
            DataSetAurora.UsersKeywordsRow userKeywordRow = getUserKeyword( keyword, userId );
            userKeywordRow.notInterestingOccurrences += keyword.Occurrences;
            usersKeywordsAdapter.Update( userKeywordRow );
        }


        #endregion

        #region get

        private static DataSetAurora.KeywordsRow getKeywordByValue( Keyword keyword )
        {
            KeywordsTableAdapter keywordsAdapter = new KeywordsTableAdapter();
            DataSetAurora.KeywordsDataTable keywordsDataTable =
keywordsAdapter.GetKeywordByValue( keyword.Value );
            return ( DataSetAurora.KeywordsRow ) keywordsDataTable.Rows[ 0 ];
        }

        public static DataSetAurora.UsersKeywordsRow getUserKeyword( Keyword keyword, string
userId )
        {
            UsersKeywordsTableAdapter usersKeywordsAdapter = new UsersKeywordsTableAdapter();
            DataSetAurora.UsersKeywordsDataTable usersKeywordsDataTable =
usersKeywordsAdapter.GetUserKeyword( keyword.Value, userId );
            return ( DataSetAurora.UsersKeywordsRow ) usersKeywordsDataTable[ 0 ];
        }

        public static List<NewsItemKeyword> getKeywordsByNewsItem( string feedUrl, string
itemUrl )
        {
            List<NewsItemKeyword> keywords = new List<NewsItemKeyword>();
```

```
            KeywordsTableAdapter keywordsAdapter = new KeywordsTableAdapter();
            DataSetAurora.KeywordsDataTable keywordsDataTable =
keywordsAdapter.GetKeywordsByNewsItem( feedUrl, itemUrl );

            foreach ( DataSetAurora.KeywordsRow keywordRow in keywordsDataTable )
            {
                string value = keywordRow.value;
                int occurrences = int.Parse( keywordRow.ItemArray[ 4 ].ToString() );

                NewsItemKeyword keyword = new NewsItemKeyword( value, feedUrl, itemUrl,
occurrences );
                keywords.Add( keyword );
            }

            return keywords;
        }

        public static bool exists( Keyword keyword )
        {
            KeywordsTableAdapter keywordsAdapter = new KeywordsTableAdapter();
            DataSetAurora.KeywordsDataTable keywordsDataTable =
keywordsAdapter.GetKeywordByValue( keyword.Value );

            if ( keywordsDataTable.Count > 0 )
                return true;
            else
                return false;
        }

        public static bool existsKeywordToUser( Keyword keyword, string userId )
        {
            UsersKeywordsTableAdapter usersKeywordsAdapter = new UsersKeywordsTableAdapter();
            DataSetAurora.UsersKeywordsDataTable usersKeywordsDataTable =
usersKeywordsAdapter.GetUserKeyword( keyword.Value, userId );

            if ( usersKeywordsDataTable.Count > 0 )
                return true;
            else
                return false;
        }

        #endregion

        public static List<UserKeyword> getUserKeywords( List<NewsItemKeyword> keywords,
string userId )
        {
            List<UserKeyword> userKeywords = new List<UserKeyword>();

            foreach ( NewsItemKeyword keyword in keywords )
            {
                DataSetAurora.UsersKeywordsRow userKeywordRow = getUserKeyword( keyword,
userId );
                UserKeyword userKeyword = new UserKeyword( keyword.Value, userId,
userKeywordRow.interestingOccurrences, userKeywordRow.notInterestingOccurrences );
                userKeywords.Add( userKeyword );
            }

            return userKeywords;
        }
    }
}
```

ControlPanelForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Aurora.Controller;
using Aurora.Types;
using Aurora.Feed.Types;


namespace ControlPanel
{
    public partial class ControlPanelForm : Form
    {
        public ControlPanelForm()
        {
            InitializeComponent();
            timerUpdateFeeds.Interval = 600000;
            textBoxUpdateInverval.Text = Convert.ToString( timerUpdateFeeds.Interval / 60000
);
          }

        private void ControlPanelForm_Resize( object sender, EventArgs e )
        {
            if ( FormWindowState.Minimized == WindowState )
            {
                Hide();
                this.notifyIconControlPanel.Visible = true;
            }
        }

        private void notifyIconControlPanel_DoubleClick( object sender, EventArgs e )
        {
            Show();
            WindowState = FormWindowState.Normal;
            this.notifyIconControlPanel.Visible = false;
        }


        private void buttonExit_Click( object sender, EventArgs e )
        {
            Close();
        }

        private void tabControlMain_SelectedIndexChanged( object sender, EventArgs e )
        {
            if ( tabControlMain.SelectedIndex == 0 )
                updateTabStatsControls();
            else
                if ( tabControlMain.SelectedIndex == 1 )
                    updateTabFeedsControls();
                else
                    if ( tabControlMain.SelectedIndex == 2 )
                        updateTabUsersControls();
        }

        private void updateTabStatsControls()
        {
```

```
        }

        private void updateTabFeedsControls()
        {
            List<NewsFeed> newsFeeds = NewsFeedController.getNewsFeeds();
            listBoxFeedsAdded.Items.Clear();

            foreach ( NewsFeed newsFeed in newsFeeds )
                listBoxFeedsAdded.Items.Add( newsFeed.FeedUrl );
        }

        private void updateTabUsersControls()
        {
            List<User> users = UserController.getUsers();
            listBoxUsersAdded.Items.Clear();

            foreach ( User user in users )
                listBoxUsersAdded.Items.Add( user.Id );

        }

        private void buttonAddUser_Click( object sender, EventArgs e )
        {
            User newUser = new User( textBoxUserName.Text );
            UserController.add( newUser );
            updateTabUsersControls();
        }

        private void buttonRemoveUser_Click( object sender, EventArgs e )
        {
            string userId = listBoxUsersAdded.SelectedItem.ToString();
            UserController.remove( userId );
            updateTabUsersControls();
        }

        private void buttonAddFeed_Click( object sender, EventArgs e )
        {
            FeedReader feedReader = new FeedReader();
            NewsFeed newsFeed = feedReader.Load( textBoxFeedUrl.Text );
            NewsFeedController.add( newsFeed );
            updateTabFeedsControls();
        }

        private void buttonRemoveFeed_Click( object sender, EventArgs e )
        {
            string feedUrl = listBoxFeedsAdded.SelectedItem.ToString();
            NewsFeedController.remove( feedUrl );
            updateTabFeedsControls();
        }

        private void listBoxUsersAdded_SelectedIndexChanged( object sender, EventArgs e )
        {
            string userId = listBoxUsersAdded.SelectedItem.ToString();
            labelFeedsUser.Text = "Feeds assinados por " + userId + ":";

            List<NewsFeed> newsFeeds = NewsFeedController.getNewsFeeds();
            List<NewsFeed> userSubscribedFeeds = NewsFeedController.getNewsFeedsByUser( userId
);

            comboBoxFeedsAvailableToUser.Items.Clear();
            foreach ( NewsFeed newsFeed in newsFeeds )
                comboBoxFeedsAvailableToUser.Items.Add( newsFeed.FeedUrl );
```

```
            listBoxFeedsSubscribedByUser.Items.Clear();
            foreach ( NewsFeed newsFeed in userSubscribedFeeds )
                listBoxFeedsSubscribedByUser.Items.Add( newsFeed.FeedUrl );
        }

        private void buttonSubscribeFeedToUser_Click( object sender, EventArgs e )
        {
            string feedUrl = comboBoxFeedsAvailableToUser.SelectedItem.ToString();
            string userId = listBoxUsersAdded.SelectedItem.ToString();

            NewsFeedController.insertNewsFeedToUser( userId, feedUrl );
            listBoxUsersAdded_SelectedIndexChanged( sender, e );
        }

        private void buttonRemoveFeedSubscriptionFromUser_Click( object sender, EventArgs e )
        {
            string feedUrl = listBoxFeedsSubscribedByUser.SelectedItem.ToString();
            string userId = listBoxUsersAdded.SelectedItem.ToString();

            NewsFeedController.removeNewsFeedFromUser( userId, feedUrl );
            listBoxUsersAdded_SelectedIndexChanged( sender, e );
        }

        private void buttonUpdateFeeds_Click( object sender, EventArgs e )
        {
            FeedReader feedReader = new FeedReader();
            NewsFeed newsFeed;

            List<string> newsFeedsUrls = NewsFeedController.getNewsFeedsUrls();
            foreach ( string url in newsFeedsUrls )
            {
                newsFeed = feedReader.Load( url );
                NewsFeedController.save( newsFeed );
            }

            labelLastUpdate.Text = DateTime.Now.ToLocalTime().ToString();
            labelNextUpdate.Text = DateTime.Now.AddMinutes( double.Parse(
textBoxUpdateInverval.Text ) ).ToLocalTime().ToString();

            updateTabStatsControls();
        }

        private void textBoxUpdateInverval_TextChanged( object sender, EventArgs e )
        {
            timerUpdateFeeds.Interval = int.Parse( textBoxUpdateInverval.Text ) * 60000;
            labelNextUpdate.Text = DateTime.Now.AddMinutes( double.Parse(
textBoxUpdateInverval.Text ) ).ToLocalTime().ToString();
        }
    }
}
```

BayesianClassifier.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Collections.Generic;
using Aurora.Feed.Types;
using Aurora.Types;
using Aurora.Feed.Types.Rss;
using Aurora.Feed.Types.Atom;
using Aurora.Controller;
```

```
namespace Aurora.Classifier
{
    public class BayesianClassifier
    {
        #region Constructor

        public BayesianClassifier()
        {
            //construtor da classe BayesianClassifier
        }

        #endregion

        #region Classifier

        public static List<NewsItem> Classify( List<NewsItem> newsItems, string userId )
        {
            User user = UserController.getUser( userId );

            foreach ( NewsItem newsItem in newsItems )
            {
                float numerator = 1;
                float denominator = 1;

                List<UserKeyword> newsItemUserKeywords = KeywordController.getUserKeywords(
newsItem.Keywords, userId );

                foreach ( UserKeyword keyword in newsItemUserKeywords )
                {
                    numerator = numerator * keyword.InterestingFrequency;
                    denominator = denominator * keyword.NotInterestingFrequency;
                }

                numerator = numerator * user.InterestingFrequency;
                denominator = numerator + ( denominator * user.NotInterestingFrequency );

                if ( denominator != 0 )
                    newsItem.Interestingness = numerator / denominator;
                else
                    newsItem.Interestingness = 0;
            }

            return newsItems;
        }

        #endregion
    }
}
```

Default.aspx.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Text;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
```

```
using System.Web.UI.HtmlControls;
using System.Collections.Generic;
using Aurora.Controller;
using Aurora.Types;
using Aurora.Feed.Types;
using Aurora.Classifier;

public partial class _Default : System.Web.UI.Page
{
    #region Properties

    protected string UserId
    {
        set
        {
            Session[ "UserID" ] = value;
        }
        get
        {
            if ( Session[ "UserID" ] == null )
                return ( null );

            return ( Session[ "UserID" ] as string );
        }
    }

    #endregion

    public static string getPostBackEventTarget( System.Web.HttpRequest request )
    {
        return ( request[ "__EVENTTARGET" ] );
    }

    public static string getPostBackEventArgument( System.Web.HttpRequest request )
    {
        return ( request[ "__EVENTARGUMENT" ] );
    }

    protected void Page_PreInit( object sender, EventArgs e )
    {
        UserId = "UsuarioA";
    }

    protected void Page_Load(object sender, EventArgs e)
    {
        if ( !this.IsPostBack )
        {
            getUnreadedNewsItems();
        }
        else
        {
            string action = getPostBackEventArgument( this.Request );

            if ( action == "GetMoreNews" )
            {
                markNewsItemsAsNotReaded();
                selectNextUnreadedNewsItems();
            }
            else
                if ( action == "CheckUpdates" )
                {
                    markNewsItemsAsNotReaded();
                    getUnreadedNewsItems();
```

```
                }
                else
                {
                    int index = int.Parse( action );
                    markNewsItemAsReaded( index );
                }
            }
        }

        private void getUnreadedNewsItems()
        {
            List<NewsFeed> userSubscribedFeeds = NewsFeedController.getNewsFeedsByUser( UserId );
            List<NewsItem> totalUnreadedNews = new List<NewsItem>();

            foreach ( NewsFeed newsFeed in userSubscribedFeeds )
                totalUnreadedNews.AddRange( NewsItemController.getUnreadedNewsItems(
newsFeed.FeedUrl, UserId ) );

            BayesianClassifier.Classify( totalUnreadedNews, UserId );
            totalUnreadedNews.Sort( delegate( NewsItem newsItem1, NewsItem newsItem2 ) { return
newsItem2.Interestingness.CompareTo( newsItem1.Interestingness ); } );

            this.Cache[ UserId + "_totalUnreadedNews" ] = totalUnreadedNews;

            selectNextUnreadedNewsItems();
        }


        private void markNewsItemsAsNotReaded()
        {
            List<NewsItem> unreadedNews = ( this.Cache[ UserId + "_unreadedNews" ] as
List<NewsItem> );

            foreach ( NewsItem newsItem in unreadedNews )
            {
                if ( !newsItem.IsReaded )
                {
                    NewsItemController.markAsReaded( newsItem, UserId );
                    KeywordController.updateKeywordsAsNotReaded( newsItem.Keywords, UserId );
                    UserController.updateNotInterestingNewsItemsCount( UserId );
                }
            }
        }

        private void selectNextUnreadedNewsItems()
        {
            List<NewsItem> totalUnreadedNews = ( this.Cache[ UserId + "_totalUnreadedNews" ] as
List<NewsItem> );
            List<NewsItem> unreadedNews = new List<NewsItem>();

            int count = 10;
            HyperLinkMoreNews.Text = "Mais notícias";
            HyperLinkMoreNews.NavigateUrl = ClientScript.GetPostBackClientHyperlink(
this.HyperLinkMoreNews, "GetMoreNews" );

            if ( totalUnreadedNews.Count < 10 )
            {
                count = totalUnreadedNews.Count;
                HyperLinkMoreNews.Text = "Verificar atualização";
                HyperLinkMoreNews.NavigateUrl = ClientScript.GetPostBackClientHyperlink(
this.HyperLinkMoreNews, "CheckUpdates" );
            }
```

```
        unreadedNews.AddRange( totalUnreadedNews.GetRange( 0, count ) );
        totalUnreadedNews.RemoveRange( 0, count );

        this.Cache[ UserId + "_unreadedNews" ] = unreadedNews;

        populateUnreadedNewsList( unreadedNews );
    }

    private void markNewsItemAsReaded( int index )
    {
        List<NewsItem> unreadedNews = ( this.Cache[ UserId + "_unreadedNews" ] as
List<NewsItem> );

        NewsItem newsItem = unreadedNews[ index ];
        newsItem.IsReaded = true;
        NewsItemController.markAsReaded( newsItem, UserId );
        KeywordController.updateKeywordsAsReaded( newsItem.Keywords, UserId );
        UserController.updateInterestingNewsItemsCount( UserId );

        populateUnreadedNewsList( unreadedNews );
    }

    private void populateUnreadedNewsList( List<NewsItem> unreadedNews )
    {
        int index = 0;

        foreach ( NewsItem newsItem in unreadedNews )
        {
            Panel panelNewsItem = new Panel();
            panelNewsItem.SkinID = "PanelNewsItem";

            HyperLink hyperLinkItemTitle = new HyperLink();
            hyperLinkItemTitle.Target = "_blank";
            hyperLinkItemTitle.NavigateUrl = newsItem.ItemUrl;
            hyperLinkItemTitle.Text = newsItem.Title;
            hyperLinkItemTitle.SkinID = "HyperLinkItemTitle";
            hyperLinkItemTitle.Attributes.Add( "onclick",
ClientScript.GetPostBackClientHyperlink( hyperLinkItemTitle, index.ToString() ) );
            panelNewsItem.Controls.Add( hyperLinkItemTitle );

            panelNewsItem.Controls.Add( new LiteralControl( "<br />" ) );

            Label labelItemPubDate = new Label();
            if ( newsItem.Published != null )
                labelItemPubDate.Text = newsItem.Published.Value.ToLocalTime().ToString();
            labelItemPubDate.SkinID = "LabelItemPubDate";
            panelNewsItem.Controls.Add( labelItemPubDate );

            Label labelFeedName = new Label();
            labelFeedName.Text = "";
            labelFeedName.SkinID = "LabelFeedName";
            panelNewsItem.Controls.Add( labelFeedName );

            panelNewsItem.Controls.Add( new LiteralControl( "<br />" ) );

            Label labelItemContent = new Label();
            labelItemContent.Text = newsItem.Content;
            labelItemContent.SkinID = "LabelItemContent";
            panelNewsItem.Controls.Add( labelItemContent );

            PanelNews.Controls.Add( panelNewsItem );
            index++;
        }
```

```
    }
}
```

Script para criação da base de dados

```
create table Users (
    userID varchar( 20 ),
    newsItemsReviewed int,
    interestingNewsItems int,
    notInterestingNewsItems int,
    constraint PK_Users primary key ( userID )
)

create table NewsFeeds (
    feedUrl varchar( 255 ),
    title varchar( 255 ),
    description varchar( 1024 ),
    language varchar( 5 ),
    imageUrl varchar( 255 ),
    copyright varchar( 100 ),
    generator varchar( 50 ),
    updated datetime,
    author varchar( 50 ),
    constraint PK_NewsFeeds primary key ( feedUrl )
)

create table UsersNewsFeeds (
    userID varchar( 20 ),
    feedUrl varchar( 255 ),
    constraint PK_UsersNewsFeeds           primary key ( userID, feedUrl ),
    constraint FK_UsersNewsFeeds_Users     foreign key ( userID )  references Users,
    constraint FK_UsersNewsFeeds_NewsFeeds foreign key ( feedUrl ) references NewsFeeds
)

create table NewsItems (
    itemUrl varchar( 255 ),
    feedUrl varchar( 255 ),
    title varchar( 255 ),
    content text,
    author varchar( 50 ),
    published datetime,
    constraint PK_NewsItems           primary key ( itemUrl, feedUrl ),
    constraint FK_NewsItems_NewsFeeds foreign key ( feedUrl ) references NewsFeeds
)

create table UsersNewsItems (
    itemUrl varchar( 255 ),
    feedUrl varchar( 255 ),
    userID varchar( 20 ),
    isReaded bit,
    constraint PK_UsersNewsItems primary key ( itemUrl, feedUrl, userID ),
    constraint FK_UsersNewsItems_NewsItems foreign key ( itemUrl, feedUrl ) references
NewsItems,
    constraint FK_UsersNewsItems_Users foreign key ( userID ) references Users
)

create table Keywords (
    keywordID int identity( 1, 1 ),
    value varchar( 50 ) not null unique,
    constraint PK_Keywords primary key ( keywordID )
)
```

```
create table NewsItemsKeywords (
    itemUrl varchar( 255 ),
    feedUrl varchar( 255 ),
    keywordID int,
    numberOccurrences int,
    constraint PK_NewsItemsKeywords           primary key ( itemUrl, feedUrl, keywordID ),
       constraint FK_NewsItemsKeywords_NewsItems foreign key ( itemUrl, feedUrl ) references
NewsItems,
    constraint FK_NewsItemsKeywords_Keywords  foreign key ( keywordID )        references
Keywords
)

create table UsersKeywords (
    userID varchar( 20 ),
    keywordID int,
    interestingOccurrences int,
    notInterestingOccurrences int,
    constraint PK_UsersKeywords          primary key ( userID, keywordID ),
    constraint FK_UsersKeywords_Users    foreign key ( userID )    references Users,
    constraint FK_UsersKeywords_Keywords foreign key ( keywordID ) references Keywords
)
```

APÊNDICE 2 – ARTIGO

Software Agregador de Notícias com Classificador Bayesiano

Daniel Fagundes da Silva

Departamento de Informática e Estatística (INE)
Centro Tecnológico (CTC)
Universidade Federal de Santa Catarina (UFSC)
Florianópolis – Santa Catarina – Brasil

danielfs@inf.ufsc.br

Abstract. *This essay describes the development of a news aggregator, capable to classify the order in which the news articles are presented to the user, accordingly with the analysis of the keywords present in each news article. It describes about the theoretical background required to make the parse and extraction of content from the most frequent feed formats available on Internet today. It is presented an overview about the basic Probability theory required to comprehend the classification process, and it is given an example of the application of the Bayes' Theorem. By the end, the development process of the news aggregator is detailed, to the bayesian classifier that it used to determine the probable user's interest degree in each news article.*

Resumo. *O presente trabalho descreve o desenvolvimento de um software agregador de noticias, capaz de classificar a ordem em que as notícias são apresentadas ao usuário, de acordo com a análise das palavras-chave presentes em cada notícia. Discorre sobre a fundamentação teórica para a análise e extração de conteúdo dos formatos de feeds mais usados na Internet. Uma visão geral da teoria básica de Probabilidade para o processo de classificação é apresentada, e um exemplo da aplicação do Teorema de Bayes é fornecido. Por fim, é detalhado o processo de desenvolvimento do software agregador de notícia e do classificador bayesiano usado para determinar o provável grau de interesse do usuário para cada notícia.*

1. Introdução

Com o aumento da oferta de notícias *online*, o usuário comum vê-se frente a frente com um volume cada vez maior de conteúdo. O problema da sobrecarga de informação reside no fato de que o leitor em busca informação por vezes não consegue encontrar a informação que deseja em meio à massa de informações que ele não considera interessante. A utilização de softwares e *websites* agregadores de notícias resolvem uma parte do problema, evitando que o usuário tenha de acessar individualmente os *websites* de sua preferência, em busca de conteúdo atualizado. Porém, mesmo a utilização destes recursos não impede que o usuário receba diariamente dezenas, por vezes centenas, de notícias das mais diversas fontes e tenha de percorrer, uma a uma, tais notícias em busca de conteúdo relevante.

A proposta do presente trabalho é desenvolver um aplicativo agregador de notícias, sob a forma de um *website*, que seja capaz de realizar a classificação das notícias trazidas da Internet, na ordem da mais relevante para a menos relevante, através da análise das palavras-chave contidas em cada notícia, calculando o grau de possível interesse por parte do usuário em uma determinada notícia, através do uso de um classificador bayesiano.

2. Teorema de Bayes

Sejam A e B dois eventos de um espaço amostral S. Considerando que a probabilidade de observar simultaneamente os eventos A e B , através do Teorema da multiplicação, é dada por $P(A \cap B) = P(B \mid A) \cdot P(A)$ ou $P(A \cap B) = P(A \mid B) \cdot P(B)$, pode-se afirmar que:

$$P(B \mid A) \cdot P(A) = P(A \mid B) \cdot P(B)$$

Rearranjando a expressão acima pode-se obter:

$$P(B \mid A) = \frac{P(A \mid B) \cdot P(B)}{P(A)}$$

A expressão acima é conhecida como *Teorema de Bayes*, também denominada de *Teorema da Probabilidade das "Causas"*. Ela permite *atualizar* uma crença na possibilidade de ocorrência de um evento B (probabilidade *a posteriori*) baseado na experiência prévia conhecida para a possibilidade de o evento B ocorrer (probabilidade *a priori*) em conjunto com a observação da ocorrência de uma evidência A a respeito do evento B [YUDKOWSKY].

Como em geral não se conhece $P(A)$, pode-se reescrever esta probabilidade sob a forma:

$P(A) = P(A \cap B) + P(A \cap B')$, onde B' é o evento complementar de B.

Através do Teorema da multiplicação, pode-se substituir $P(A \cap B)$ e $P(A \cap B')$ na expressão acima e obter:

$$P(A) = P(A \mid B) \cdot P(B) + P(A \mid B') \cdot P(B')$$

Substituindo então no Teorema de Bayes a expressão para $P(A)$ estabelecida acima, obtém-se a formulação alternativa para o Teorema de Bayes:

$$P(B \mid A) = \frac{P(A \mid B) \cdot P(B)}{P(A \mid B) \cdot P(B) + P(A \mid B') \cdot P(B')}$$

Por fim, considerando os eventos B_1, B_2, ..., B_n como sendo mutuamente exclusivos, formando uma partição do espaço amostral S, é possível elaborar o Teorema de Bayes sob a forma:

$$P(B_i \mid A) = \frac{P(A \mid B_i) \cdot P(B_i)}{P(A \mid B_1) \cdot P(B_1) + P(A \mid B_2) \cdot P(B_2) + \ldots + P(A \mid B_n) \cdot P(B_n)}$$

A expressão acima define que desde que os B_i constituam uma partição do espaço amostral, um e somente um dos eventos B_i ocorrerá. Portanto, a expressão acima nos dá a probabilidade de um evento particular B_i ter ocorrido (ou seja, uma "causa") dado que o evento A ocorreu [MEYER].

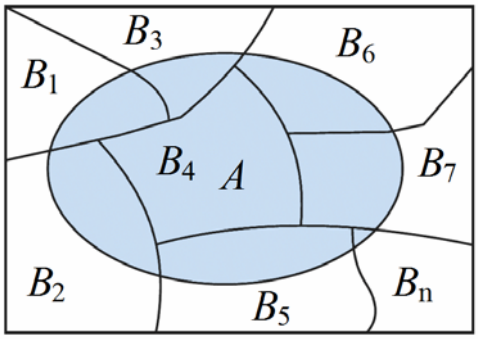

Figura 1. Eventos B_i formando uma partição do espaço amostral S.

3. Metodologia

3.1 Visão Geral

O agregador proposto foi desenvolvido de visando facilitar o acompanhamento de *feeds* (com enfoque em *feeds* de notícias) de forma a personalizar a ordem de apresentação das notícias para um usuário de acordo com o monitoramento da utilização passada da aplicação por parte deste usuário.

Utilizando uma técnica aplicada anteriormente a filtros bayesianos para a eliminação de SPAMs (correio eletrônico não-solicitado e comumente indesejável) em caixas postais de correio eletrônico, o software agregador procura antecipar o interesse do usuário em certas notícias através da análise do conteúdo da notícia, atribuindo à ela um valor, correspondente ao possível grau de interesse do usuário, utilizado então para classificar a ordem em que as notícias são apresentadas.

3.2 Aplicação do Teorema de Bayes

O exemplo a seguir propõe-se a explicar a forma na qual o Teorema de Bayes é utilizado para a classificação de notícias no presente trabalho. Utilizando a combinação de novas evidências acerca de um evento, calcula-se a probabilidade de que este evento tenha ocorrido. Neste caso, deseja-se calcular a probabilidade de uma notícia ser considerada interessante pelo usuário baseado no conteúdo presente na notícia.

Como forma de simplificação do problema e facilitar o cálculo das probabilidades, assume-se a independência da ocorrência entre as palavras encontradas em cada notícia. Tal abordagem é conhecida na literatura como *Classificador Bayesiano Ingênuo* (ou *Naive Bayes Classifier*) [DOMINGOS] [RISH] e apesar de independência entre as palavras não ser comumente verdadeira, este classificador mostra-se surpreendentemente efetivo, quando comparado a outros métodos de aprendizado de máquina mais complexos [VINOKOUROV] [ZHANG].

Para a aplicação desta abordagem no problema de classificação de notícias, fez-se uma primeira distinção entre o grau de interesse (ou desinteresse) do usuário em relação às notícias a serem analisadas, estabelecendo duas categorias possíveis: notícias *interessantes* e *não-interessantes*.

O ato de leitura, por parte do usuário, de uma determinada notícia, classifica-a como interessante para aquele usuário. Notícias não-lidas são consideradas como não-interessantes. Desta forma, para o universo de todas as notícias passíveis de serem apresentadas ao usuário, os eventos de uma noticia ser considerada interessante ou não-interessante são mutuamente

exclusivos, formando uma partição deste universo. Assim, considerando novas evidências para uma determinada notícia (as palavras-chave que representam seu conteúdo), pode-se atualizar a probabilidade da notícia ser considerada *interessante* dada a observação dos eventos de ocorrência das palavras-chave presentes na notícia.

Considerando *X, Y* e *Z* o universo de três palavras possíveis de ocorrer em uma determinada noticia, pode-se simplificar, para fins de compreensão, o modelo proposto para classificação da seguinte forma (considerações para um usuário qualquer):

Número de notícias consideradas **interessantes**: 40

Número de notícias consideradas **não-interessantes**: 60

Número total de notícias avaliadas: 100

Notícias / Palavras	**X**	**Y**	**Z**
Ocorrência em notícias **interessantes**	20	8	16
Ocorrência em notícias **não-interessantes**	6	54	18

Notícias / Palavras	**X**	**Y**	**Z**
Frequência em notícias **interessantes**	50%	20%	40%
Frequência em notícias **não-interessantes**	10%	90%	30%

Através destas informações, é possível aplicar a fórmula do Teorema de Bayes para calcular o provável grau de interesse deste usuário quando surgirem novas notícias contendo uma ou mais destas palavras.

Considerando *I* o evento de uma notícia ser interessante e *I*' seu evento complementar, para a *palavra X* tem-se que:

$$P(I \mid X) = \frac{P(X \mid I) \cdot P(I)}{P(X \mid I) \cdot P(I) + P(X \mid I') \cdot P(I')} = 0{,}7692$$

Calculando o restante das probabilidades de interesse para as demais combinações possíveis obtém-se:

Provável grau de interesse		
P (interesse | X)	0,7692	76,92%
P (interesse | Y)	0,1290	12,90%
P (interesse | Z)	0,4706	47,06%
P (interesse | X, Y)	0,4255	42,55%

P (interesse | X, Z)	0,8163	81,63%
P (interesse | Y, Z)	0,1649	16,49%
P (interesse | X, Y, Z)	0,4969	49,69%

A utilização do agregador atualiza constantemente as freqüências das palavras-chave em notícias interessantes e não-interessantes. Com isto é possível "treinar" a aplicação de acordo com a utilização passada de cada usuário, calculando o possível grau de interesse para cada notícia nova trazida.

3.3 Escolha das palavras-chave

Nem todas as palavras presentes em uma notícia são essenciais para a determinação de seu significado. Como forma de aprimorar o processo de cálculo do grau de interesse da notícia, palavras que apresentam um baixo conteúdo informacional, denominadas de *stopwords*, são removidas do conteúdo indexado no banco de dados.

Este processo de remoção de *stopwords* implica o fato de que algumas palavras possuem um peso maior de significância do que outras, para o processo de avaliação do conteúdo do texto [SOUZA]. Para tanto, houve a necessidade de criação de uma lista de *stopwords* a serem desconsideradas quando presentes em uma notícia. Esta lista, denominada de *stoplist*, é composta na maioria das vezes por termos que não identificam um contexto específico, sendo inerentes à linguagem e ao idioma e não ao conteúdo da notícia.

Através do estudo da gramática da língua portuguesa, selecionaram-se classes gramaticais que caracteristicamente compõem uma *stoplist*.

3.4 Módulos da aplicação

A aplicação de forma geral consiste em dois módulos principais, que executam de forma independente. O primeiro módulo, denominado *Painel de controle*, é um software executável, para o sistema operacional Windows, desenvolvido para alimentar uma base de dados com notícias dos *feeds* cadastrados no sistema. O segundo módulo consiste em uma interface *web*, denominada *Aurora*, que se conecta na base de dados alimentada pelo *Painel de controle* e busca notícias ainda não lidas de *feeds* assinados por um usuário.

Em suporte a estes dois módulos encontra-se uma biblioteca de classes desenvolvida para realizar a análise, conversão e extração de palavras-chave de *feeds* e notícias, e um módulo classificador, que utiliza um classificador bayesiano para tentar antecipar o interesse do usuário em determinadas notícias.

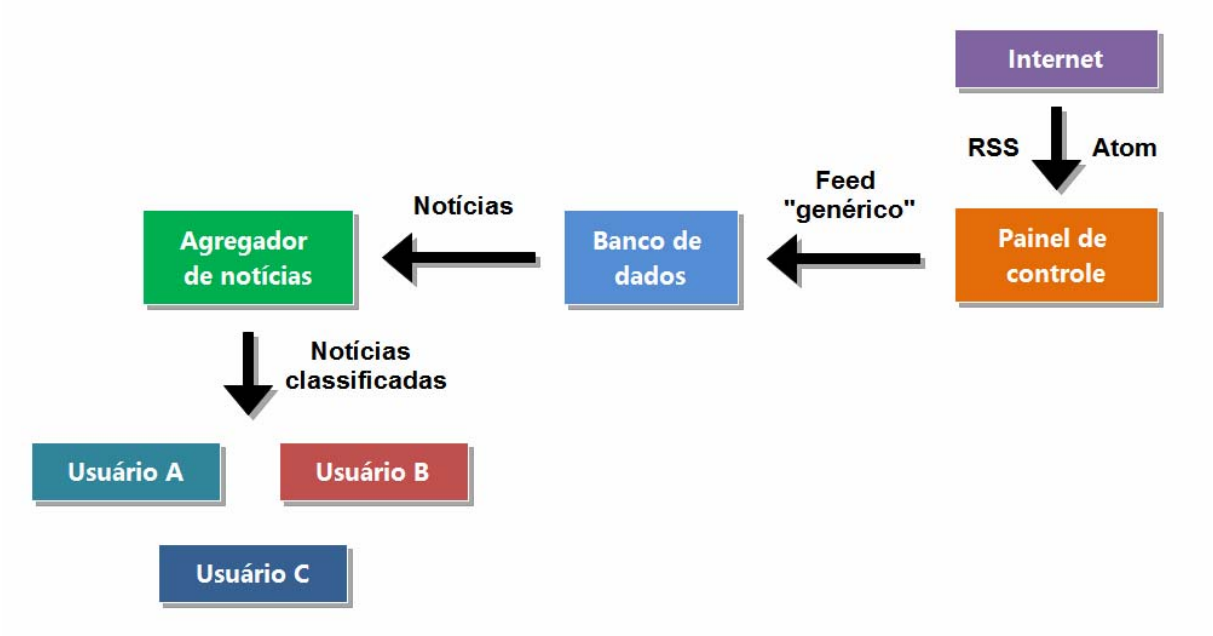

Figura 2. Visão geral da aplicação.

O software *Painel de controle* é responsável por buscar novas notícias de forma automática, para *feeds* cadastrados no sistema. As notícias são então convertidas para um formato de *feed* genérico usado pela aplicação e salvas no banco de dados.

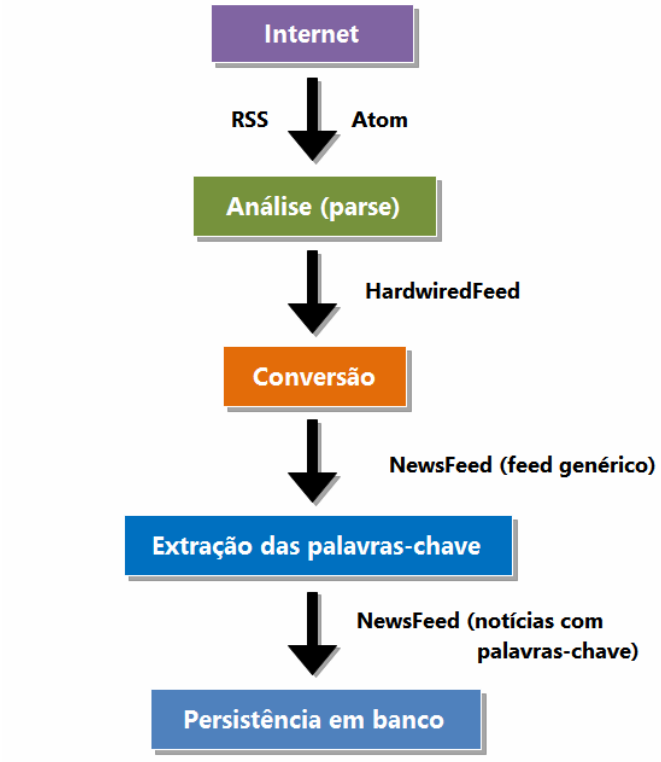

Figura 3. Atividades do *Painel de controle*.

3.5 Análise

O primeiro passo realizado no processo de análise consiste em buscar o conteúdo do *feed* solicitado. Este pode estar localizado tanto na Internet quanto em um arquivo acessível através da máquina na qual o software *Painel de controle* está sendo executado.

Após o conteúdo do *feed* ter sido carregado, seu *stream* XML é passado para a classe que realiza a análise do conteúdo do *feed*. O processo de análise (*parse*) de um *feed* consiste primeiramente em analisar o cabeçalho do XML passado e identificar o formato e a versão do *feed*. A classe de análise então realiza a chamada do método de *parse* da classe responsável por aquele tipo especifico de *feed*. Cada uma destas classes conhece a lógica por trás da especificação do formato tratado por ela. Portanto, são elas as responsáveis por extrair as informações do stream XML trazido da Internet.

As informações extraídas por cada classe de análise são inseridas em objetos que representam *feeds* de formatos específicos. Após a análise de um *feed* RSS, não importando a versão, a classe responsável pela análise feita devolve um objeto representando os atributos comuns de um *feed* RSS; de forma similar, após a análise de um *feed* Atom, um objeto específico para este formato de *feed* é retornado.

3.6 Conversão

Como mencionado, os objetos devolvidos pelas classes de análise ainda são atrelados à formatos específicos e não correspondem à idéia de um formato genérico de *feed*, necessário para trafegar informação entre as camadas da aplicação. Com isto torna-se necessário convertê-los para um formato de *feed* genérico, de forma a melhor gerenciar as informações trazidas de diferentes fontes de distribuição de conteúdo e em diferentes formatos.

O processo de conversão é feito por classes que conhecem a estrutura por trás de cada formato de *feed* específico. Desta forma, através da análise dos atributos presentes no objeto de *feed* específico, um formato de *feed genérico* é criado e seus atributos preenchidos, sendo devolvido para o uso interno da aplicação em substituição ao objeto de formato específico.

3.7 Extração das palavras-chave

Um terceiro passo é necessário para tornar o objeto de *feed* genérico útil para sua utilização no processo de classificação das notícias que ocorre momentos antes de serem exibidas ao usuário. Este passo consiste na extração das palavras-chave consideradas relevantes em cada notícia.

Cada *feed* no formato genérico possui como atributo uma lista de notícias, também em formato genérico. Cada objeto que representa uma notícia, por sua vez, possui como atributo uma lista de objetos que representam palavras-chave.

O processo de extração de palavras-chave de uma notícia, de forma geral, consiste primeiramente em normalizar seu conteúdo através da remoção de elementos indesejáveis ao processo de análise do texto. Em seguida as palavras consideradas irrelevantes são removidas através do uso de uma *stoplist*. Por fim, as palavras restantes são adicionadas, uma a uma, como objetos que representam as palavras-chave para uma determinada notícia. Cada notícia processada é então devolvida com sua lista de palavras-chave preenchida, pronta para ser armazenada no banco de dados.

3.8 Classificador bayesiano

O processo de classificação de uma notícia, leva em consideração evidências (palavras-chave) presentes em cada notícia a ser classificada. Com os valores de frequência de cada palavra-chave em notícias interessantes e não-interessantes sendo atualizados a todo instante no banco de dados, a partir da utilização da aplicação por parte do usuário, o processo de classificação das notícias deve ocorrer momentos antes das mesmas serem disponibilizadas para a leitura do usuário.

A classe responsável pela classificação de uma lista de notícias recebe como argumento uma lista de notícias a ser classificada. O primeiro passo do classificador é obter as freqüências de ocorrência de cada palavra-chave, para cada notícia, em notícias interessantes e não-interessantes. Portanto, para cada notícia, o classificador solicita do banco de dados estes valores de frequência em relação ao usuário atual, recebendo-os como uma lista de palavras-chave para o usuário. A seguir, o classificador solicita a frequência total de notícias interessantes e não-interessantes avaliadas pelo usuário atual. De posse destes valores, o classificador pode então, através da aplicação do Teorema de Bayes, calcular o possível grau de interesse do usuário baseado nas palavras-chave presentes na notícia. Palavras-chave presentes mais de uma vez são consideradas de acordo com sua frequência na notícia, elevando ou diminuindo a probabilidade da notícia ser considerada interessante.

Após ter sido obtido o valor da probabilidade, o classificador insere este valor *dentro* da notícia sendo avaliada e o processo se repete para todas as outras notícias presentes no *feed*. Ao final do processo, o classificador devolve o mesmo *feed* passado como argumento, porém com os valores do possível grau de interesse de cada notícia presente em todas elas. É interessante notar que o classificador não ordena as notícias; ele apenas atribui a elas um valor específico que pode então ser usado posteriormente para alterar a maneira de apresentar as notícias de acordo.

4. Implementação

4.1 Painel de controle

O software *Painel de controle* desenvolvido tem o propósito de alimentar uma base de dados com notícias trazidas da Internet. O usuário final da aplicação não interage diretamente com este módulo da aplicação, sendo seu uso reservado para o administrador responsável pelo agregador de notícias.

O *Painel de controle* é uma aplicação Windows que executa em background em uma máquina servidor, conectando-se à Internet a cada intervalo de tempo pré-determinado, em busca de novas notícias para todos os *feeds* assinados por usuários do sistema. Para cada *feed* cadastrado, o *Painel de controle* utiliza a classe responsável pela leitura dos formatos de *feeds* trazidos da Internet, passando como parâmetro a URL do *feed* que necessita ser atualizado. A biblioteca de classes para o tratamento de *feeds* então se responsabiliza por conectar-se à Internet e manipular o stream XML do *feed*, devolvendo ao *Painel de controle* um objeto de *feed* genérico com o conteúdo necessário.

Para cada objeto de *feed* recebido, o *Painel de controle* atualiza o banco de dados com as novas notícias presentes em cada *feed* analisado. Desta forma, sempre que um usuário solicita a leitura de novas notícias, garante-se um intervalo máximo de defasagem entre a distribuição do conteúdo *online* e sua inclusão no banco de dados de notícias.

4.2 Agregador de notícias

A implementação principal do trabalho consiste no agregador de notícias, responsável por exibir as notícias não lidas para um determinado usuário. Optou-se pelo desenvolvimento de uma interface simplificada para o agregador, com todas as notícias não lidas de todos os *feeds* assinados por um usuário sendo exibidas como uma lista única.

O agregador consiste em uma página *web*, que exibe uma listagem de dez notícias de cada vez. É dada ao usuário a possibilidade de recarregar a lista com novas notícias não lidas, até que todas as notícias disponíveis para leitura em um determinado momento sejam mostradas.

O agregador é o responsável pela ordenação das notícias de acordo como foram classificadas. A página que exibe as notícias para o usuário solicita ao banco de dados uma lista de objetos de *feed* genérico, contendo todas as notícias não lidas para todos os *feeds* assinados pelo usuário. A seguir, todas as notícias não lidas são reunidas em uma única lista que é então passada para a classe que realizará a classificação bayesiana das mesmas.

Quando recebe de volta a lista de notícias com suas respectivas probabilidades, a página *web* do agregador ordena a lista de acordo com os valores calculados para cada notícia. Esta é considerada pela aplicação a ordem ideal para apresentação das notícias ao usuário. Por fim, de posse da lista de notícias ordenadas, o agregador exibe-as para o usuário e então aguarda, esperando receber o *feedback* da leitura (ou não) das notícias mostradas.

O ato de leitura de uma notícia dispara um evento de *postback* da página *web* que exibe as notícias. Neste momento, no servidor, todas as palavras-chave pertencentes à notícia lida têm sua frequência em notícias interessantes atualizadas e em seguida uma nova janela do navegador é aberta, exibindo o conteúdo da notícia para o usuário.

De forma análoga, cada vez que o usuário solicita a exibição de mais notícias, todas as palavras-chave de todas as notícias não-lidas que estavam sendo mostradas têm seus valores de frequência em notícias não-interessantes atualizados e então uma nova leva de dez notícias não lidas é mostrada.

Este processo repete-se durante a utilização do agregador pelo usuário e os valores de frequência em notícias interessantes e não interessantes vão sendo atualizados, para uma posterior classificação feita para novas notícias que sejam adicionadas ao banco de dados.

5. Conclusão

A utilização do teorema de Bayes se mostrou uma solução viável quando aplicado ao problema de classificação de notícias. O baixo custo de implementação do algoritmo aliado aos poucos parâmetros necessários para obter um valor com o qual se pode inferir um possível interesse do usuário em uma determinada notícia mostraram-se ideais para utilização computacional do teorema em uma aplicação prática.

Entretanto, um aspecto importante que se destacou posteriormente no trabalho foi a importância na escolha das palavras-chave a serem utilizadas no processo de classificação. Optou-se por um método simples de eliminação de palavras-chave desnecessárias, através da utilização de *stopwords*. Apesar de satisfatório, a utilização somente de *stopwords* pode não garantir que o processo de escolha de palavras-chave seja o mais adequado. Uma análise posterior de outros métodos de extração de palavras-chave em *corpus* de notícias faz-se necessária para garantir a qualidade do processo de classificação.

Referências

DOMINGOS, P.; PAZZANI, M. J. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss, Kluwer Academic Publishers - Hingham, MA, USA., v. 29, p. 103-130, 1997.

MEYER, P. L. Probabilidade: aplicações à estatística. 2. ed. Rio de Janeiro: LTC – Livros Técnicos e Científicos Editora S.A., 1983.

RISH, I. An empirical study of the naive Bayes classifier, T.J. Watson Research Center - Hawthorne, NY, USA.

SOUZA, R. R. Uma proposta de metodologia para escolha automática de descritores utilizando sintagmas nominais. Belo Horizonte, 2005. 215f. Tese apresentada ao Programa de Pós-Graduação em Ciência da Informação da Universidade Federal de Minas Gerais como requisito parcial à obtenção do título de Doutro em Ciência da Informação – Escola de Ciência da Informação, Universidade Federal de Minas Gerais.

VINOKOUROV, A. The Organisation and Retrieval of Document Collections: A Machine Learning Approach. Paisley, Scotland, 2003. 213 f. Dissertation submitted in partial fullfilment of the requirements for the Degree of Doctor of Philosophy – School of Information and Communication Technologies, University of Paisley.

YUDKOWSKY, E. An Intuitive Explanation of Bayesian Reasoning. Disponível em: <http://yudkowsky.net/bayes/bayes.html/> Acesso em: 03, jun. 2007.

ZHANG, H.; SU, J. Naive Bayesian Classifiers for Ranking. In: EUROPEAN CONFERENCE ON MACHINE LEARNING (ECML2004), 15., 2004, Pisa, Italy. Proceedings of…, Springer, 2004.