APRIMORAMENTO E REFATORAÇÃO DO SISTEMA COPIA E COLA

Luiz Felipe Nazari[1]

RESUMO

O Copia e Cola é um sistema *on-line* para a detecção de plágio desenvolvido por professores e acadêmicos da Universidade do Oeste de Santa Catarina (Unoesc) de Xanxerê desde 2010. Nesse período, a aplicação tem apresentado uma crescente demanda pelas pesquisas proporcionalmente ao número de usuários, levando à necessidade de aprimoramento da ferramenta, em termos de usabilidade, eficiência, *feedback* ao usuário e, principalmente, qualidade dos resultados das pesquisas. Como objetivo inicial, visou-se à ampliação da gama de tipos de arquivos manipulados pela aplicação, após isso, à refatoração do sistema tanto em sua estrutura quanto nos códigos-fonte. Para atingir os objetivos, foram realizados estudos bibliográficos, leitura de artigos disponíveis na internet, estudo dos padrões de formatação de textos e parágrafos, diversas experimentações e pesquisas de ferramentas. As implementações realizadas resultaram em uma menor quantidade de problemas relatados por usuários, melhor usabilidade quanto à submissão e visualização de arquivos, resultados mais precisos pela manipulação textual e maior nível técnico da aplicação, sendo flexível a mudanças por utilizar recursos avançados de programação orientada a objetos. O aumento de usuários ativos (maioria constituída por pessoas com vínculos a instituições de ensino) reforça a importância do sistema e da detecção de plágio.
Palavras-chave: Sistema *web*. Detecção. Plágio.

1 INTRODUÇÃO

A crescente demanda pelas pesquisas de detecção de plágio pela aplicação para *web* Copia e Cola (COPIA E COLA, 2016) implicou a necessidade do aprimoramento da ferramenta, melhorando sua usabilidade, *feedback* ao usuário e, principalmente, a ampliação dos tipos de arquivos aceitos e a qualidade dos resultados das pesquisas.

Até o momento da elaboração deste artigo, o Copia e Cola continha, em sua base de dados, um total de 47.391 usuários cadastrados. A maioria dos usuários é constituída por pessoas que possuem algum vínculo com instituições de ensino (destacam-se estudantes universitários e professores). Esses dados foram obtidos por meio do *feedback* obtido, da análise dos endereços de *e-mail* cadastrados e dos nomes de arquivos enviados à pesquisa de plágio (destacam-se artigos, monografias e relatórios de estudos) e do preenchimento da instituição de ensino nos cadastros da aplicação.

Ao analisar a estrutura do sistema, foram observados vários pontos precários que poderiam ser aprimorados (como organização e padronização de projeto e dos códigos-fonte) e outros que deveriam ser mais bem explorados (como o poder da programação orientada a objetos). O projeto mostrou-se difícil de ser entendido e mantido, reforçando a necessidade de refatoração.

Outro ponto analisado para a alteração se refere à ampliação dos arquivos aceitos para a pesquisa. Em suas primeiras versões, o sistema apenas aceitava arquivos nos formatos .doc e .docx, extensões dos arquivos gerados pelo programa Microsoft Word para o sistema operacional Windows da Microsoft, fato que limitava (ou dificultava) a utilização por usuários de outros sistemas. Na versão vigente a gama de arquivos aceitos foi ampliada com a inclusão das extensões .pdf e .txt, estas utilizadas amplamente por qualquer sistema e que atenderam às necessidades dos usuários.

Ao trabalhar apenas com os formatos .doc e .docx surgiu um novo problema: a dificuldade de visualizar os arquivos-resposta gravados pelo sistema. Por serem gerados a partir do documento original, o formato era mantido, prendendo-se ao Microsoft Word. Os endereços de *sites* com possibilidade de plágio misturavam-se ao texto original, dificultando a e desvirtuando o foco da leitura das referências encontradas.

Além das alterações referentes aos arquivos, uma nova lógica de gerenciamento de pesquisa foi implementada com fins de aumentar a capacidade de busca do sistema. A maior dificuldade foi em relação às limitações dos buscadores

[1] Graduado em Tecnologia em Análise e Desenvolvimento de Sistemas pela Universidade do Oeste de Santa Catarina; luiz.nazari.42@gmail.com

(serviços de busca *on-line*) utilizados pelo Copia e Cola, os quais apresentam limites na quantidade de pesquisas em um espaço de tempo (impedindo a realização de pesquisas simultâneas, em que uma interferiria na outra).

2 **FUNDAMENTAÇÃO TEÓRICA**

2.1 PROGRAMAÇÃO ORIENTADA A OBJETOS

A orientação a objetos é um paradigma da programação que trabalha com classes e objetos (abstrações de conceitos do mundo real), definindo comportamentos e relacionamentos. Diferente do paradigma procedural, no qual instruções bem definidas são seguidas à risca, "[...] em linguagens orientadas a objeto, a implementação [...] é fundamental, mas pensar no projeto de classes, em como elas se encaixam e como elas serão estendidas é o que importa." (ANICHE, 2015, p. 1-2).

Conforme Caelum (2016, p. 34), as principais vantagens da orientação a objetos são maior organização e legibilidade, além concentrar e encapsular as lógicas de negócio nos pontos certos e tornar o código flexível em razão da capacidade de polimorfismo entre classes.

2.2 REFATORAÇÃO

A refatoração consiste em "[...] uma técnica na qual o código é reestruturado de forma a não haver modificação de seu comportamento [...] Sendo assim, o código vai sendo alterado em pequenos passos até que se chegue ao padrão-alvo." (GUERRA, 2013, p. 18). Em projetos existentes, a aplicação de padrões é feita com base na técnica da refatoração.

2.3 PADRÃO *STRATEGY*

Conforme Guerra (2013, p. 14), o *Strategy* "[...] é um padrão que deve ser utilizado quando uma classe possuir diversos algoritmos que possam ser utilizados de forma intercambiável." É caracterizado pela utilização de heranças e interfaces, ou seja, conceitos que abrangem o polimorfismo. A partir da estrutura criada para uma classe, novas implementações podem ser introduzidas posteriormente, sem a alteração do código existente.

3 **METODOLOGIA**

Para a resolução deste trabalho foram realizadas pesquisas bibliográficas explorando os conceitos abordados e suas aplicabilidades. Como o problema teve origem em um projeto com códigos-fonte já implementados, a técnica de refatoração para o emprego de padrões de desenvolvimento foi adotada, como proposto por Guerra (2013).

A elaboração das soluções foi possível utilizando a Linguagem de Modelagem Unificada (UML) que, conforme Guerra (2013, p. 238), é uma ferramenta para a representação dos modelos de classes mediante diagramas, para auxiliar o programador a visualizar o problema e o relacionamento de classes e, dessa forma, alcançar uma solução antes da implementação.

Utilizando essas estratégias, conduziram-se análises lógicas e estruturais sobre o código-fonte do sistema, levantando os pontos necessários de alterações, para, então, definir como seguiria o desenvolvimento nas questões de onde e como aplicar as abstrações, otimizações e relacionamento entre classes de forma a beneficiar-se pelo polimorfismo. Tais aspectos são considerados por Aniche (2015) como a parte mais complexa da programação orientada a objetos.

Neste trabalho, cada ponto identificado para a alteração está descrito nas seções dos elementos referentes ao desenvolvimento.

4 **DESENVOLVIMENTO**

4.1 ADAPTAÇÃO DO SISTEMA PARA RECEBER ALTERAÇÕES

No início de seu desenvolvimento[1] [2]o Copia e Cola não seguia padrões de projeto e implementação. Logo, a aplicação de padrões, assim como a alteração das rotinas então desenvolvidas foram realizadas com base na técnica da refatoração.

Por mais que a linguagem Java seja uma linguagem com suporte à programação orientada a objetos, os principais processos do sistema encontravam-se implementados de forma procedural, regados a comandos condicionais, dificultando a legibilidade e tornando o código muito mais dispendioso de ser mantido (características de lógicas procedurais).

As primeiras alterações realizadas foram referentes à organização e à padronização do projeto Java, assim como à nomenclatura dos pacotes e classes. O estudo de recursos da programação orientada a objetos fez-se necessário nessa etapa, definindo a estrutura de classes, os relacionamentos entre elas e a melhor forma de utilização nos processos do sistema.

Em decorrência do fato de o Copia e Cola trabalhar com a manipulação de vários formatos de arquivos e gerenciar vários buscadores, o padrão *Strategy* foi aplicado para definir a estrutura das classes às funcionalidades. Assim, foram criadas classes abstratas e interfaces para generalizar o comportamento do código, permitindo que suas implementações possam ser trocadas em tempo de execução, adotando comportamento dinâmico.

4.2 LEITURA DE ARQUIVOS

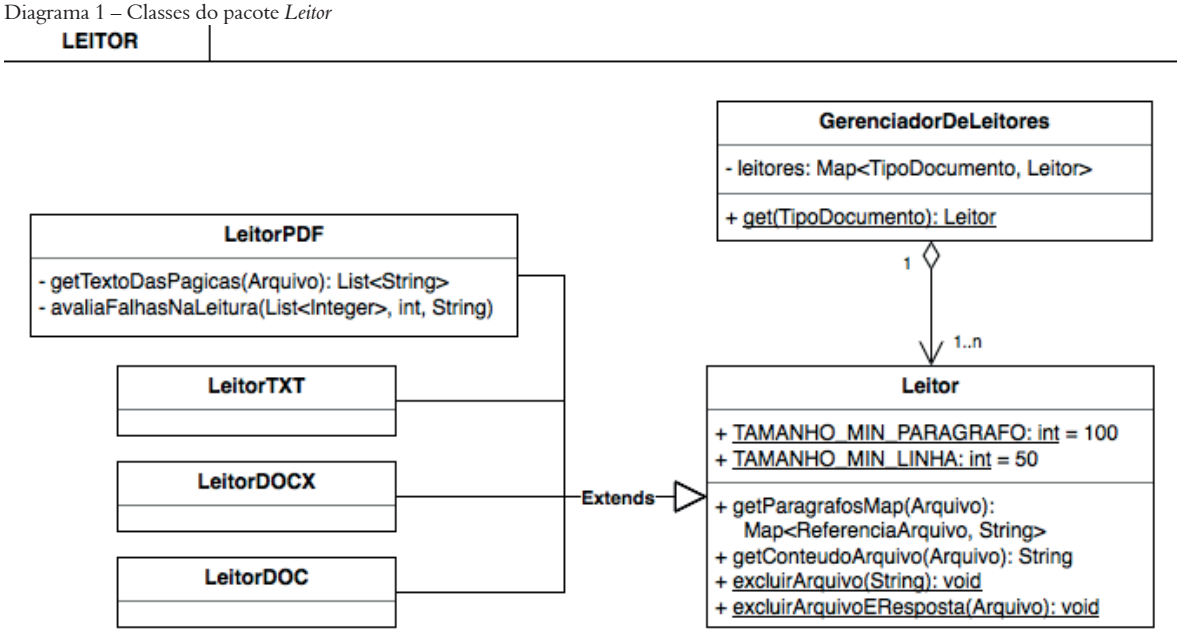
A leitura de arquivos para a identificação de parágrafos e frases presentes nos documentos enviados para a análise é de fundamental importância. Os arquivos aceitos nas primeiras versões do sistema (.doc e .docx) limitavam o uso a usuários do programa Microsoft Word para o sistema operacional Windows da Microsoft.

Sendo um dos processos mais complexos do sistema, são necessárias duas etapas para a efetuação da leitura: a extração dos textos e a identificação dos trechos relevantes à pesquisa. Essas etapas são de responsabilidade dos leitores de arquivo, bem como a estrutura de classes específicas por reconhecer os tipos de documentos e devem ser capazes de extrair e filtrar seus conteúdos.

As rotinas de leitura de documentos estavam misturadas ao código de extração de plágio, assim como parte do tratamento dos textos e a decisão de qual leitor utilizar, geração do arquivo-resposta e construção e envio do e-mail contendo o resultado, em meio a regras de decisões complexas "se, senão-se, senão".

Essas rotinas foram então separadas em um pacote e classes específicas. O Diagrama de classes da figura 1 demostra as novas classes do pacote *Leitor*:

Diagrama 1 – Classes do pacote *Leitor*



Fonte: o autor.

[2]O projeto Copia e Cola foi iniciado no ano 2010 contando com o Professor orientador e idealizador do Software Cristiano Agosti, e desenvolvido pelo então acadêmico do Curso de Ciências da Computação da Universidade do Oeste de Santa Catarina, Luzitan Orso Zancan.

Ao estender a classe *Leitor* (homônima ao pacote), as classes adotarão um comportamento genérico, categorizando o polimorfismo. Cada extensão de *Leitor* possui as regras específicas para a extração de textos de cada tipo de arquivo.

Quadro 1 – Exemplo de código utilizando as classes de leitura de arquivo

```
Arquivo arquivo = new Arquivo();
Leitor leitor = GerenciadorDeLeitores.get(arquivo.getTipoDocumento());
leitor.getParagrafosMap(arquivo);
```

Fonte: o autor.

No Quadro 1 pode-se visualizar um exemplo prático da utilização dos leitores. A implementação do leitor utilizada dependerá do tipo do arquivo: quando for enviado um arquivo .txt, será utilizado o *LeitorTXT* e, ao enviar um arquivo .docx, o *LeitorDOCX*.

Dadas as refatorações, para desenvolver e incluir um novo leitor "especializado" de arquivo, basta criar uma nova classe que estenda de *Leitor* e, então, adicioná-la no *GerenciadorDeLeitores* para poder ser utilizada, sem a necessidade de alterar uma única linha das rotinas de pesquisa do sistema.

Realizando esse processo, foram criados os leitores com suporte às extensões de arquivos .pdf e .txt, e, posteriormente, .rtf e .odt, com os já existentes, eliminando os problemas de usabilidade de usuários referentes aos arquivos enviados. O Copia e Cola passou a não receber mais reclamações por incompatibilidades de arquivos, atendendo às necessidades da grande maioria dos usuários.

4.3 GRAVAÇÃO DE ARQUIVOS E GERAÇÃO DO RESULTADO DA PESQUISA

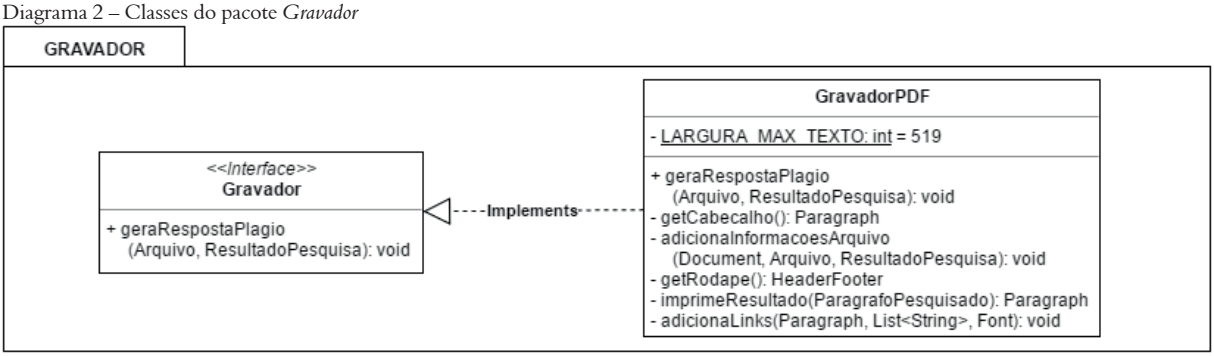
Em razão do fato de que o sistema trabalhava apenas com arquivos .doc e .docx para a leitura, isso se repetia aos documentos-resposta, os arquivos gravados pelo sistema contendo o resultado da pesquisa de plágio.

Os endereços de *sites* com possibilidade de plágio encontrados em um parágrafo ou frase eram anexados após o respectivo trecho no arquivo original, misturando-se ao texto e aos demais elementos presentes. A leitura dos endereços dos *sites* tornava-se difícil em decorrência da poluição visual, uma vez que os endereços encontrados se misturavam às formatações originais dos textos. O mesmo problema da leitura repetia-se: dificuldade em visualizar os documentos para usuários que não utilizavam o sistema Microsoft Word.

Com o intuito de resolver o problema, estudou-se a possibilidade de padronizar o documento-resposta e, com o suporte a documentos PDF, aproveitar-se da usabilidade deles, uma vez que esse formato de arquivo pode ser lido até mesmo nos navegadores de internet comumente utilizados, além de melhorar a legibilidade, pois o foco é mantido apenas nos resultados.

A implementação no código ocorreu de forma semelhante aos leitores. Dessa vez, foi utilizada uma interface para padronizar os gravadores de arquivos.

Diagrama 2 – Classes do pacote *Gravador*



Fonte: o autor.

Com a estrutura demonstrada no Diagrama 2, há uma generalização oferecida pela interface *Gravador*, e uma única implementação, o *GravadorPDF*, cuja responsabilidade é construir um documento real contendo os parágrafos detectados com possíveis plágios e seus respectivos endereços de *sites*, em que seus conteúdos foram comparados como iguais ou parecidos.

A presente implementação solucionou a dificuldade de leitura das respostas por parte de usuários, e dificuldades de visualização, não necessitando de outras opções. Entretanto, caso um outro formato venha a ser desejado, basta criar uma implementação.

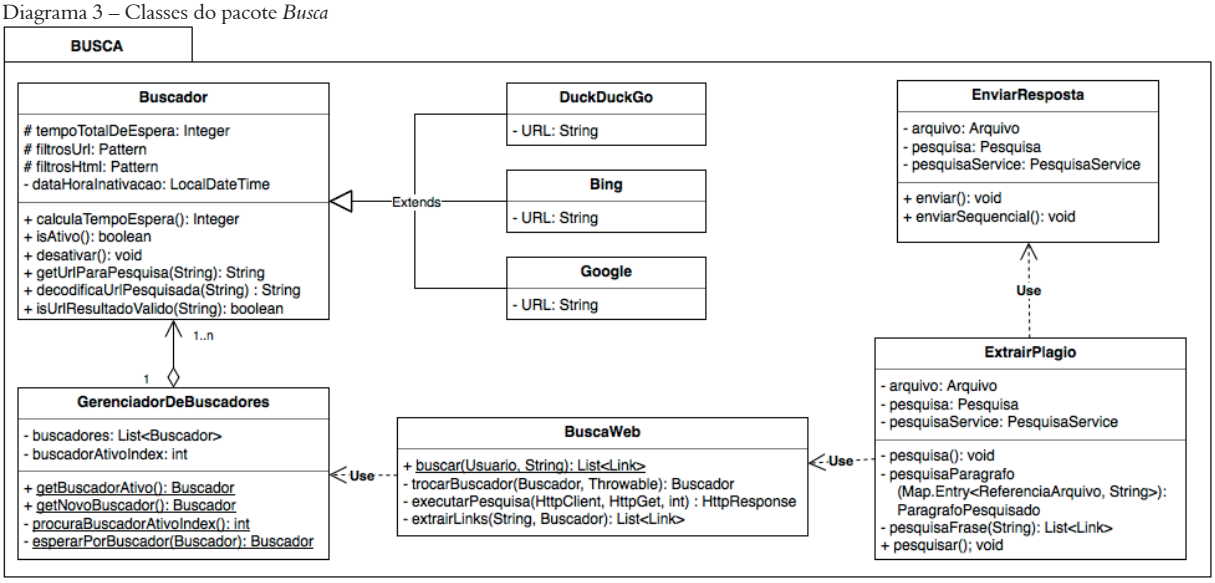
4.4 PROCESSO DE BUSCA POR PLÁGIO

A busca por plágio é núcleo do sistema, no qual estão concentrados todos os processos para torná-la possível. As rotinas que envolvem a busca podem ser separadas em quatro etapas principais: a leitura de arquivos de texto, os algoritmos de busca por referências em conteúdos *on-line* e o gerenciamento de mecanismos de buscas, a geração do documento resposta e, finalmente, o envio da resposta ao usuário.

Todas essas rotinas estavam misturadas, compondo o "supercódigo" de extração de plágio. Tudo estava acoplado na classe *ExtrairPlagio*.

Por motivos claros, a classe *ExtrairPlagio* foi inteiramente refatorada, tendo seu algoritmo segregado em diferentes e específicas classes, compondo o pacote *Busca*. Todas suas classes, funcionalidades e relacionamentos podem ser visualizados no Diagrama 3:

Diagrama 3 – Classes do pacote *Busca*



Fonte: o autor.

Como o Copia e Cola utiliza mecanismos de buscas de terceiros para realizar a detecção de plágio em documentos *on-line* (atualmente estão em produção os buscadores referentes aos sites Google, DuckduckGo e Bing e sendo testados Ask e UolBusca), faz-se necessário o gerenciamento dos buscadores.

É no *GerenciadorDeBuscadores* que são gerenciadas as especificações da abstração *Buscador* demonstradas no Diagrama 3. É a classe responsável por selecionar qual buscador será usado para as pesquisas. Cada buscador possui limitação da quantidade de pesquisas por uma faixa de tempo e um tempo de "resfriamento" até poder ser utilizado novamente, por isso, é importante o controle de tempo de espera e o horário de inativação do serviço.

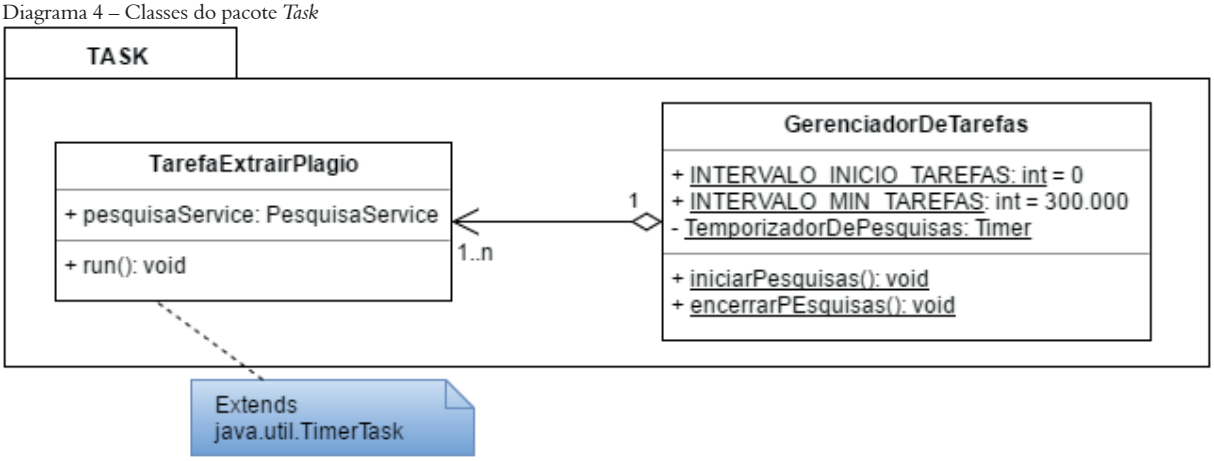
Aproveitando o polimorfismo, cada característica particular de um buscador, como a forma de busca na *web*, o tempo de espera, o tratamento de URLs e os resultados, pôde ser implementada em sua respectiva classe, alterando, assim, o comportamento do mecanismo de busca conforme a implementação provida pelo gerenciador.

4.4.1 **Gerenciamento de tarefas de buscas por plágio**

As rotinas responsáveis por controlar os processos de busca também tiveram sua implementações refatoradas. Em seus princípios, o sistema possuía controle dos processos por *Threads* (uma *Thread* corresponde a um processo computacional) com a ideia de poder utilizar de múltiplas *Threads* de pesquisas. Entretanto, em razão das limitações dos buscadores, esse comportamento não foi possível. Apenas uma *Thread* era executada por vez e todas as pesquisas pendentes eram carregadas no início do sistema e também alimentadas pela submissão *on-line* de arquivos.

O sistema acabava, por fim, com vários processos em memória e apenas um em execução, não utilizando a capacidade de *Multithreading* e gerenciamento. Para resolver o problema de memória, foram estudadas as *Tasks* (tarefas) com um gerenciador.

Diagrama 4 – Classes do pacote *Task*



Fonte: o autor.

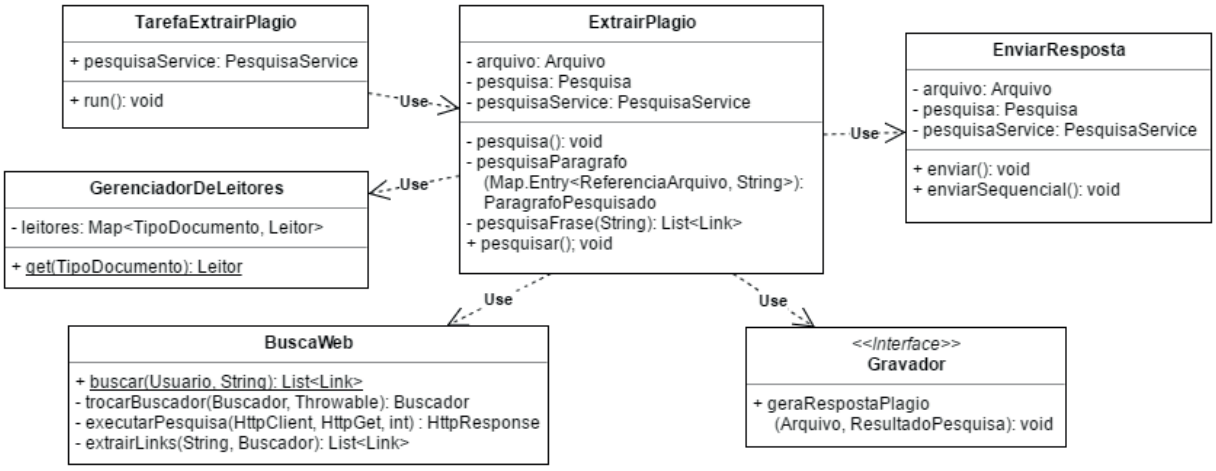
O Diagrama 4 ilustra o pacote *Task*, em que o *GerenciadorDeTarefas* é responsável por iniciar tarefas e controlar o tempo mínimo de início de execução e a classe *TarefaExtrairPlagio* implementa o método run() herdado da classe *TimerTask* (nativa do Java), a qual busca uma pesquisa pendente (por ordem de prioridade e tempo de submissão) no banco de dados e cria uma instância de *ExtrairPlagio*, executando, por fim, uma pesquisa.

Para eliminar a necessidade de verificação de novas pesquisas a cada segundo, o gerenciador especifica um tempo mínimo de execução entre tarefas (estipulado em cinco minutos). Caso o tempo de execução de uma tarefa for maior do que o mínimo e houver outra tarefa pendente, esta será iniciada imediatamente após o término da atual.

4.4.2 **Integração das funcionalidades para o processo de extração de plágio**

Como descrito na seção de gerenciamento de tarefas, a principal rotina executada faz referência à classe *ExtrairPlagio*. Essa é a classe responsável por unir todas as pontas da execução central do sistema na realização de buscas por plágio, possuindo, dessa forma, dependência das demais classes elucidadas neste artigo. O Diagrama 5 representa um diagrama de classes que ilustra a integração da classe *ExtrairPlagio*.

Diagrama 5 – Integração das funcionalidades entre os pacotes para o processo de extração de plágio



Fonte: o autor.

Essencialmente, o processo central do sistema consiste nos seguintes passos:

a) uma *TarefaExtrairPlagio* é iniciada, criando uma instância de *ExtrairPlagio* e, posteriormente, invoca seu método público pesquisar();

b) a instância de *ExtrairPlagio* demanda uma implementação de *Leitor* ao *GerenciadorDeLeitores* para extrair o conteúdo do arquivo a ser pesquisado;

c) os parágrafos lidos são decompostos em frases que serão filtradas e repassadas ao método estático buscar() da classe *BuscaWeb* a qual utilizará o *Buscador* para obter as listas de referências de possíveis plágios aos parágrafos pesquisados;

d) a relação de parágrafos e referências de possíveis plágios é repassada para uma implementação de *Gravador* para gerar o arquivo-resposta;

e) a tarefa é encerrada iniciando o envio do *e-mail* ao usuário (assincronamente) contendo o arquivo-resposta e a próxima tarefa é iniciada, repetindo o processo.

5 **CONCLUSÃO**

Durante o início do projeto foram percebidas dificuldades nas questões de entendimento e manutenção dos algoritmos existentes. O sistema apresentava problemas de usabilidade que limitavam a utilização por usuários e resultados confusos em meio ao arquivo de texto original enviado à pesquisa de plágio.

Após a refatoração das principais rotinas do sistema, foi percebido um grande salto no nível técnico e na manutenibilidade da aplicação, tornando-a de mais fácil compreensão (em nível de código-fonte) em razão da padronização estrutural e flexível a mudanças por utilizar recursos avançados de programação orientada a objetos. A implementação de novas rotinas tornou-se um processo claro e fácil de ser realizado.

Com a inclusão do suporte às extensões de arquivo .pdf e .txt e, posteriormente, .rtf e .odt, e a padronização do documento-resposta, os problemas de usabilidade foram resolvidos e a melhor manipulação dos parágrafos extraídos dos arquivos possibilitaram a obtenção de resultados cada vez mais precisos e com maior rapidez em decorrência do gerenciamento dos serviços de busca utilizados.

Além disso, o grande número de usuários com vínculos a instituições de ensino ativos na aplicação reflete a importância do sistema e da detecção de plágio e o poder da ferramenta em apresentar uma potência considerável à divulgação e à publicidade nessa área.

Em alternativa às limitações dos buscadores, uma sugestão para a continuidade do sistema Copia e Cola seria a modularização dos atuais pacotes do projeto Java, seguindo o conceito de *Software As A Service* (*SAAS*) (Programas

Como Um Serviço, em tradução direta) permitindo a oferta de serviços a clientes interessados,[3] ao exemplo dos algoritmos de extração e dos filtros de conteúdo dos arquivos e, principalmente, o serviço central de buscas por referências de possíveis plágios. O processamento paralelo (distribuído) das pesquisas entre vários clientes também poderia ser uma solução, uma vez que a limitação dos buscadores é referente à máquina em que o serviço (de busca) foi utilizado.

Improvement and code refactoring of the Copy and Paste system

Abstract

Copia e Cola (Copy and Paste) is a web application that detects plagiarism from text files. This program has been developed by teachers and students of Universidade do Oeste de Santa Catarina (Unoesc) of Xanxerê since 2010. During this period, the application has shown a growing demand for research, in proportion to the number of users. Therefore, it denotes need of tool improvement, in terms of usability, efficiency, feedback, and, primarily, the accuracy of the results. As the main goal, we aimed the expansion of file formats that can be manipulated by the application, and the refactoring of the system structure and source code. To achieve these objectives, researches on how can text elements patterns be defined, reading of books and internet articles and various experimentations were conducted. The implementations made resulted in fewer problems related by users, better usability due the ease in submitting and viewing files, more accurate results due to textual manipulation and improvements on the technical level of the application, which become flexible to feature changes for using advanced resources of object oriented programming. The raising number of active users (majority constituted by people related to educational institutes) strengthens the importance of the application and plagiarism detection.
Keywords: Web application. Detection. Plagiarism.

REFERÊNCIAS

ANICHE, M. **Orientação a Objetos e Solid. Para Ninjas: Projetando Classes Flexiveis**. 1. ed. São Paulo: Casa do Código, 2015. 148 p.

CAELUM. **FJ-11 - Java e Orientação a Objetos**. E-Book. 2016. Disponível em: <https://www.caelum.com.br/apostila-java-orientacao-objetos/>. Acesso em: 10 abr. 2016.

COPIA E COLA. Disponível em: <http://www.copiaecola.com.br>. Acesso em: 10 abr. 2016.

GUERRA, E. **Design Patterns com Java – Projeto Orientado a Objetos Guiado por Padrões**. 1. ed. São Paulo: Casa do Código, 2013. 251 p.

[3]O Copia e Cola já recebeu contato de usuários sugerindo versões do sistema em que poderiam realizar as próprias buscas, ou ainda, ter prioridades na análise dos arquivos enviados.