

# **Relatório Final de Trabalho**

## **Arquitetura de Software**

**Pedro Araujo Pires**

Uma linha de produtos de software (SPL, em inglês) é uma forma de se criar uma família de produtos, baseando-se em *features* que podem ou não estar presentes no programa final. Esse programa é feito escolhendo-se quais features devem estar presentes. Tradicionalmente, esse trabalho é feito utilizando-se diretivas de pré-processamento (`#ifdef`). Porém, essa estratégia deixa o código confuso, de difícil compreensão. Para resolver esse problema, foi proposto utilizar cores para diferenciar o código de *features* do código do *core* do programa.

O CIDE+ é um plugin para a IDE Eclipse, que permite a visualização por cores das features de um programa. Além disso, também é possível exportar o código, selecionando quais features devem estar no código final. O CIDE+ é baseado em um outro plugin que implementa a mesma funcionalidade, o CIDE. A diferença é que o CIDE+ possui um algoritmo para a coloração semi-automática das features. O usuário escolhe uma semente (pacote, classe ou método), e todas as referências a essas sementes são marcadas no código.

Este trabalho foi iniciado em cima de uma versão já implementada do CIDE+, e o objetivo é implementar novas funcionalidades para que ele seja transformado em um produto final. A primeira funcionalidade é a possibilidade de se marcar manualmente features no código, pois o algoritmo de coloração semi-automática não é perfeito, e o usuário precisa de marcar trechos de código manualmente. Outra funcionalidade é a possibilidade de se

esconder as cores no editor (modo “light”), pois quando muitas features estão sendo mostradas na tela, o editor fica muito colorido, sendo visualmente ruim de se trabalhar no código.

O CIDE+ possui quatro pacotes, que são: *automation*, *model*, *UI* e *utils*. O pacote *automation* contém a lógica do algoritmo de extração semi-automática. Apesar de os padrões estabelecidos para escolha de módulos seja separar o código de interface do usuário do *core* do programa, esse módulo também possui o código relacionado às caixas de diálogo onde o usuário escolhe as *seeds* do algoritmo. Essa configuração já estava pronta quando eu comecei a trabalhar com o código, e como as tarefas a serem desenvolvidas eram outras, isso não foi modificado.

O pacote *model* possui o modelo de features utilizado pelo programa. Esse pacote não possui nenhuma dependência das bibliotecas do eclipse, podendo ser utilizado por outro programa sem a necessidade de alteração de código. Também não sofreu modificações pois não era o objetivo deste trabalho.

O pacote *UI* possui a maior parte do código de interface com o usuário (a exceção é o código na pacote *automation*). A interface consiste em um *popup menu* na na visualização de pacotes, através do qual é possível interagir com o *plugin*, uma visualização da AST onde é possível ver quais nós pertencem à cada feature, e os menus e ações utilizados no editor. A visualização da AST é um recurso poderoso na hora de navegar pelo programa visualizando o que pertence à cada feature, mas devido a problemas na hora de executar a marcação manual de features, ela foi desabilitada. **Falar mais quais são esses problemas**

Por fim, o pacote *utils* contém duas classes utilitárias, sendo a classe `ASTUtils` responsável por métodos para manipular o modelo das features, e a classe `PluginUtils` responsável por métodos relacionados ao ambiente do eclipse. A classe que manipula o modelo possui métodos de persistência das features, assim como métodos para fazer o mapeamento entre um trecho do código selecionado e um nó da AST. A segunda classe utilitária, que manipula o ambiente do eclipse, é responsável por encontrar qual o arquivo sendo editado no editor, a qual projeto esse arquivo pertence, e pegar qual pedaço de texto

está selecionado. Essa classe proporcionou um grande aprendizado sobre o ambiente do eclipse.

A primeira tarefa realizada foi a implementação da marcação de features direto no editor. Essa funcionalidade é necessária pois existem elementos nos programas que, mesmo fazendo parte de uma *feature*, não estão diretamente relacionados com nenhuma *seed*, fazendo com que eles não sejam marcados.

A marcação manual de features funciona da seguinte forma: primeiro o usuário seleciona um pedaço de código. Ao clicar com o botão direito do mouse sobre a área do editor, é aberto um menu onde existe a opção de o usuário escolher a qual *feature* aquele trecho de código pertence. Quando o usuário escolhe uma *feature*, o trecho de código selecionado é marcado como pertencente àquela *feature*. Porém, o usuário não pode marcar um pedaço arbitrário de código, pois dessa forma muitos erros de sintaxe poderiam ser introduzidos no código quando uma variante do programa for gerada. Esses erros de sintaxe podem ocorrer caso um nó da AST fosse marcado parcialmente, fazendo com que ele ficasse incompleto.

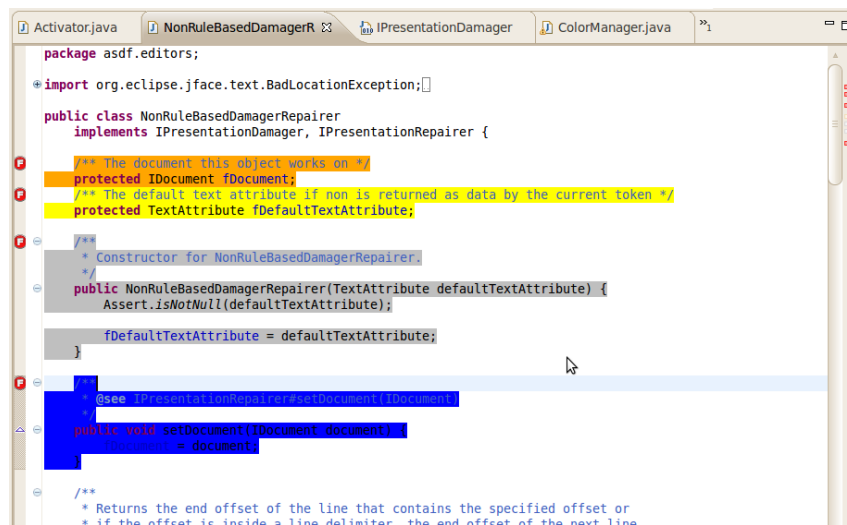
A solução para o problema foi fazer com que as marcações sejam sempre em cima da AST do programa, ao invés de serem direto no texto, garantindo que as marcações sejam sempre de acordo com a sintaxe da linguagem java. Para fazer o mapeamento entre marcação do texto e nó da AST foi utilizada a classe utilitária `ASTUtils`.

Para minimizar os erros dos usuários na hora de selecionar o texto no editor, na hora de marcar a seleção como parte de uma *feature*, é verificado se o texto selecionado corresponde a um nó da AST. Caso corresponda, não há problemas e o nó é marcado. Caso não corresponda, é buscado na AST do programa o nó mais interno que engloba todo o texto selecionado. Se o nó encontrado possuir comentários logo acima (local padrão dos comentários explicativos), os comentários também são adicionados à *feature*.

A segunda tarefa feita foi a implementação da visualização das features nas barras laterais do eclipse. Essa visualização faz parte do modo “*light*” do CIDE+, e faz com que seja mais fácil visualizar onde as *features* estão no código. Essa implementação foi feita utilizando

a API de *markers* e *annotations* do eclipse. Os *markers* são a forma disponibilizada pelo eclipse para se marcar trechos relevantes de texto, e as *annotations* são a forma de visualizar esses *markers*. Essa implementação foi relativamente simples, pois a API de *markers* é simples de usar, e possui uma extensa documentação na web.

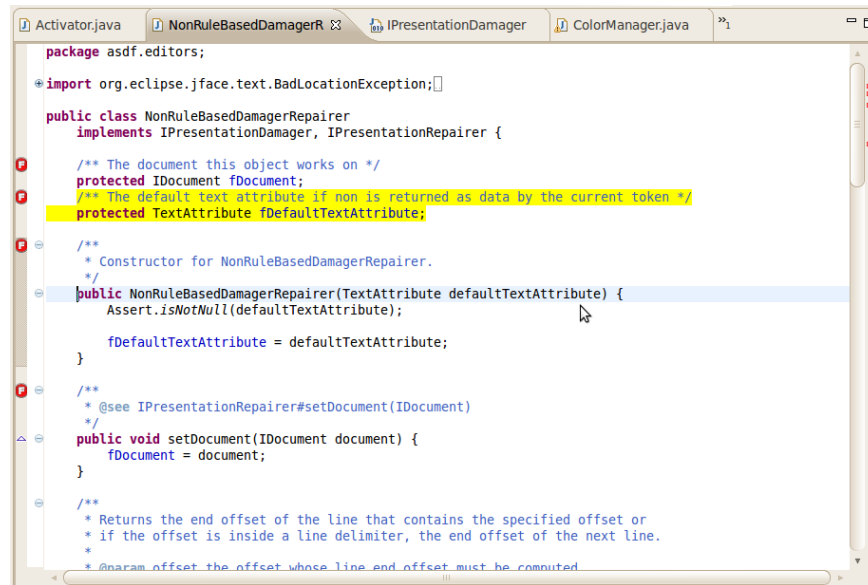
Devido ao fato de as *features* serem sempre marcadas diretamente na AST do programa, esta era lida, e a cada nó encontrado que fosse parte de alguma *feature*, o trecho correspondente de código era colorido no editor. Quando havia algum erro de sintaxe no código, a AST não era gerada, e o código não era colorido. Para resolver esse problema foi criado um novo *painter* (classe responsável pela apresentação do texto). Esse *painter* utiliza os *markers* criados para cada *feature*, uma vez que cada *marker* armazena informações como a qual *feature* ele pertence, em qual linha está o trecho de código que ele referencia, e o tamanho desse trecho. Essa implementação ainda possui uma vantagem, que é o fato de o eclipse automaticamente atualizar as informações de posição dos *markers* quando o texto é editado, de forma que o *painter* não necessitou de nenhum código adicional para atualizar a apresentação do texto enquanto este é editado.



CIDE+ com modo “light” desligado

A terceira tarefa feita foi a implementação da possibilidade de se esconder as cores no editor. Esse é a funcionalidade principal do modo “light”. Através do menu no visualizador de pacotes é possível desabilitar o *painter* responsável pela apresentação do texto. Quando as cores estão desabilitadas, as marcações na barra lateral ainda são mostradas. Se o usuário

clicar em alguma dessas marcações, a *feature* relacionada à marcação clicada é colorida, e se ocorrer um novo clique, a marcação é descolorida. A coloração da *feature* com o clique é importante pois as marcações na barra lateral não possuem informações sobre o tamanho do trecho de código pertencente à *feature*, de forma que sem as cores não é possível saber a extensão do código de *feature*.



CIDE+ com modo “light” ligado

Foram resolvidos também alguns *bugs* de interface do usuário no linux. Na caixa de diálogo que é aberta quando a extração semi-automática é executada, os botões e caixas de texto estavam sobrepostos. Foi necessário alguns ajustes na posição desses elementos para que ficassem corretos tanto no windows quanto no linux.

Atualmente estou trabalhando na melhoria do modo “light” do CIDE+. A classe responsável por capturar os clique do usuário na barra lateral ainda possui alguns *bugs*, e a marcação lateral não está satisfatória. Essa marcação atualmente é representada por uma imagem de cor fixa, de forma que não há distinção entre as marcações de duas *features* que são representadas por cores diferentes. Para resolver esse problema será implementado um sistema para gerar as imagens, sendo uma para cada *feature*.

Durante a implementação da modo “light”, houveram muitas dificuldades durante a implementação durante a implementação da classe que captura os cliques do usuário na barra lateral. A API fornecida pelo eclipse para a tarefa não funcionou, e para resolver o

problema a API do AWT foi utilizada diretamente, de forma que uma funcionalidade que deveria ter sido provida pelo Eclipse teve de ser reimplementada.

Outras dificuldades também apareceram devido ao fato de o CIDE+ possuir um editor que é uma extensão do editor Java padrão. Não é recomendado estender o editor do eclipse pois ele pode sofrer modificações, fazendo com que uma possível manutenção seja necessária a cada novo release do eclipse. Foi feita então uma refatoração do código, e o editor foi retirado do sistema. Todas as funcionalidades que eram providas por ele passaram a ser incorporadas ao editor java, que é configurado quando é aberto. Essa refatoração possibilitou uma melhora na manutenibilidade do código, além de proporcionar um grande aprendizado sobre a implementação dos editores no Eclipse.

Foram encontrados alguns *bugs* na implementação inicial do CIDE+. Devido ao fato de as features serem marcadas na AST, quando o trecho de código a ser marcado for um parâmetro de um método com mais de um parâmetro, não é possível marcar a virgula que delimita os parâmetros como parte da feature. O resultado é um erro de sintaxe no código exportado, quando for escolhida uma configuração onde a *feature* não está presente.

Um outro *bug* encontrado ocorre devido a inconsistências entre o arquivo que guarda as configurações de features em disco, e a configuração no momento. Mais especificamente, quando uma feature é deletada do modelo, e o algoritmo é executado, ele procura por essa feature, e como não encontra, uma exceção é lançada. Para resolver esse problema é necessário garantir a sincronização entre os arquivos em disco que guardam as features de cada projeto e a configuração atual de features.

Um outro problema encontrado ocorre quando um nó da AST marcado como parte de alguma *feature* é modificado pelo usuário. Apesar de o Eclipse mostrar corretamente a coloração no código, essa *feature* é retirada do modelo (na prática ela ainda está lá, mas como está relacionada com um nó da AST que não existe mais, ela é ignorada). Para resolver esse problema deve ser feito um sistema para manter a apresentação em sincronia com o modelo.

Devido às muitas dificuldades encontradas durante a implementação do modo “light”,

não foi possível resolver todos os problemas descritos anteriormente. Porém, como o trabalho será continuado após o término da disciplina, será possível resolvê-los para que o CIDE+ se torne uma boa ferramenta para manipular SPLs.