Visualizando a Evolução de Software com EvolTrack

Rafael da Silva Viterbo de Cepêda

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto de Matemática da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Apresentado por:

_____
Rafael da Silva Viterbo de Cepêda

Aprovado por:

_____
Profª. Cláudia Maria Lima Werner, D.Sc.
(Presidente)

_____
Prof. Leonardo Gresta Paulino Murta, D.Sc.
(Co-orientador)

_____
Isabella de Almeida da Silva, M.Sc.

_____
Prof. Eber Assis Schmitz, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
JUNHO DE 2008

Aos meus pais, meu irmão, e à Deus.

Agradecimentos

Aos meus pais e meu irmão, meus maiores exemplos de vida.

A Deus por me dar forças e perseverança.

Á Profª Cláudia Werner, que sempre acreditou no meu trabalho e sempre me apoiou nos bons e difíceis momentos desta caminhada. Você tem minha eterna gratidão.

Aos Professores Leonardo Murta e Aline Vasconcelos, que também muito me ajudaram e aconselharam durante todo este período.

A todos os colegas e professores da graduação, que me proporcionaram conhecimento e oportunidade.

À Roberta, por sua paciência e compreensão ao longo de todos estes meses de elaboração desta monografia.

Aos meus grandes amigos, que sempre estiveram comigo e que sempre acreditaram em mim.

RESUMO

Visualizando a Evolução de Software com EvolTrack

Rafael da Silva Viterbo de Cepêda

Orientadores: Cláudia Maria Lima Werner e Leonardo Gresta Paulino Murta

A tecnologia de software está sendo utilizada cada vez mais em uma ampla variedade de áreas de aplicação, freqüentemente produzindo sistemas maiores em tamanho e complexidade. Adicionalmente, em função de sua ampla penetração no cotidiano das pessoas, as atividades de manutenção e controle de evolução de software tornam-se vitais para a difusão e permanência desta tecnologia. Técnicas de visualização de software podem ser aplicadas com o objetivo de reduzir esta complexidade de entendimento. Neste contexto, a abordagem aqui proposta visa construir um mecanismo capaz de resgatar o ciclo de evolução de um projeto de software e fornecer um meio de visualização para a história destas evoluções.

ABSTRACT

Visualizing Software Evolution with EvolTrack

Rafael da Silva Viterbo de Cepêda

Supervisor: Cláudia Maria Lima Werner and Leonardo Gresta Paulino Murta

The software technology has been used more in a wide diversity of application areas, often producing systems bigger in size and complexity. Moreover, because of its wide use in the people everyday life, activities such as software maintenence and evolution control become vital to the spread and permanence of this technology. Visualization techniques can be applied aiming to reduce the understanding complexity associated. In this context, the proposed approach aims to build a mechanism capable of retrieving the evolution cycle associated to a software project and to provide a means of visualization to this evolution history.

Sumário

Capítulo 1. Introdução 11

1.1. Motivação 11

1.2. Objetivos 12

1.3. Organização do texto 15

Capítulo 2. Revisão da Literatura 17

2.1. Introdução 17

2.2. GASE 17

2.3. EvoLens 19

2.4. CVSscan 22

2.5. softChange 26

2.6. Augur 29

2.7. Outras abordagens 32

2.8. Comparação entre as ferramentas 33

2.9. Conclusão 36

Capítulo 3. Abordagem Proposta: EvolTrack 38

3.1. Introdução 38

3.2. Visão Geral 40

3.3. *Divide et impera* 41

3.3.1. Conectores (Problemas P1 e P4) 42

3.3.2. UML (Problema P2) 44

3.3.3. Formato Externo => Formato Interno (Problema P3) 45

3.3.4. Padrão de Projeto: Observer (Problema P6) 46

3.3.5. Formato Interno => Formato Externo (Problema P5) 47

3.4. Modelo Conceitual 49
3.5. Conclusão 51
Capítulo 4. Projeto e Implementação 52
4.1. Introdução 52
4.2. Infra-estrutura Eclipse 52
4.3. EvolTrack-Kernel 55
4.4. EvolTrack-LH 57
4.5. EvolTrack-Eclipse 59
4.6. Rastreando a Evolução 61
4.7. Conclusão 71
Capítulo 5. Conclusões 73
5.1. Contribuições 73
5.2. Limitações 75
5.3. Trabalhos Futuros 76
Referências Bibliográficas 78

Sumário de Figuras

Figura 1. Uma visão de utilização da abordagem proposta..... 13

Figura 2. Exemplo de diagrama gerado pelo GASE. 19

Figura 3. A ferramenta EvoLens com ponto focal no módulo jvision. 21

Figura 4. Colorações utilizadas: status da linha (a), tipo de construção (b) e autor (c). ... 23

Figura 5. Visão geral da ferramenta CVSscan (Fonte: VOINEA et al. (2005)) 25

Figura 6. Arquitetura da ferramenta softChange. 27

Figura 7. Exemplo de um gráfico gerado pelo softChange. 29

Figura 8. Tela da ferramenta Augur. 31

Figura 9. A ferramenta GEVOL sendo utilizada..... 33

Figura 10. Principais elementos participantes da abordagem. 40

Figura 11. Esquema de utilização de conectores..... 43

Figura 12. Processo de transformação do formato externo para o formato interno. 46

Figura 13. Implementação do padrão de projeto Observer na abordagem..... 47

Figura 14. Ilustração do processo de conversão do formato interno para um diagrama... 48

Figura 15. Modelo de Classes Conceitual da abordagem. 50

Figura 16. Esquema que representa resumidamente a plataforma Eclipse. 53

Figura 17. Arquivo MANIFEST.MF utilizado pelo plug-in EvolTrack-Kernel. 54

Figura 18. Arquivo PLUGIN.XML utilizado pelo plug-in EvolTrack-Kernel. 55

Figura 19. Diagrama de Classe UML com as principais classes do EvolTrack-Kernel. .. 56

Figura 20. Diagrama de Classe UML com as principais classes do EvolTrack-LH. 58

Figura 21. Diagrama de classe UML com as principais classes do EvolTrack-Eclipse. .. 60

Figura 22. Exemplo de utilização de dois monitores com o sistema Lighthouse. 62

Figura 23. Abertura da perspectiva EvolTrack no Eclipse..... 63

Figura 24. Estado do projeto após a crição de um pacote 64
Figura 25. Exemplo de utilização do sistema EvolTrack. 65
Figura 26. Código fonte da classe “FTP”. 66
Figura 27. Estado do projeto após a criação da classe “FTP” e de sua codificação. 67
Figura 28. Nova classe incluída no projeto, “Disco”. 68
Figura 29. Atributo “espaço” criado na classe “Disco”. 68
Figura 30. Método “retornaEspaco” adicionada à classe “Disco”. 69
Figura 31. Principais áreas e funcionalidades da interface com usuário do EvolTrack. ... 70

Sumário de Tabelas

Tabela 1. Quadro comparativo entre as ferramentas analisadas. 35

Tabela 2. Quadro comparativo entre as abordagens pesquisadas 74

Capítulo 1. Introdução

1.1. Motivação

Indubitavelmente, a tecnologia de software está sendo utilizada cada vez mais em uma ampla variedade de áreas de aplicação. Desta forma, seu correto funcionamento torna-se essencial para o sucesso do negócio envolvido e para a segurança do ser humano (ISO, 2005). Neste contexto, as atividades de manutenção e controle da evolução de software emergem como duas das principais áreas do ciclo de vida de um sistema computacional (PRESSMAN, 2006).

Entretanto, a natureza intangível e invisível do software, alinhado com seu crescente grau de complexidade, torna tais atividades uma tarefa de difícil gerenciamento e execução (BALL e EICK, 1996). As técnicas de visualização de software podem ser utilizadas para proporcionar um meio pelo qual desenvolvedores possam visualizar, identificar e entender os diferentes artefatos de um software, auxiliando, desta forma, as atividades supracitadas.

Uma das vantagens da visualização é que esta transfere do sistema cognitivo para o sistema perceptivo a carga despendida para uma determinada tarefa, aproveitando, assim, a habilidade do ser humano no reconhecimento de padrões e estruturas em informações visuais (ROBERTSON et al., 1993). Portanto, o emprego desta técnica torna mais fácil e natural a tarefa de compreensão da informação apresentada. Uma vez entendendo a estrutura e o funcionamento dos artefatos do software, a execução da atividade de manutenção pode se tornar uma tarefa de menor custo e complexidade.

Em relação ao controle da evolução do software, temos em boa parte dos projetos repositórios que armazenam tais informações. Isto é, em geral, este tipo de informação já

se encontra disponível. Porém, o entendimento destas informações, com o passar do tempo, sobrecarrega os desenvolvedores em suas atividades, já que a representação disponibilizada por estes repositórios muitas vezes aparece de forma textual e pouco intuitiva. Desta forma, pode-se aplicar as mesmas técnicas de visualização apresentadas anteriormente para suprir esta demanda.

Com isso, pode ser observado que a construção de abordagens para atender estes requisitos de visualização de um projeto, ao longo do seu ciclo de evolução representa um desafio interessante e, ao mesmo tempo, extremamente útil no contexto do processo de desenvolvimento de software.

1.2. Objetivos

Em face do exposto anteriormente, este trabalho visa construir um mecanismo capaz de resgatar o ciclo de evolução de um projeto de software, esteja este em andamento ou já finalizado. No segundo caso, informações históricas do projeto serão utilizadas. Além disto, a abordagem proposta deve proporcionar uma forma de visualização temporal da evolução do projeto identificada previamente. Isto é, este trabalho deve oferecer a possibilidade de navegação e visualização das características de um determinado projeto ao longo de toda sua vida.

Note que, neste caso, a característica do software que será visualizada será a sua estrutura, ou seja, os elementos que compõem o software e suas relações. Entretanto, é importante ressaltar que existem ainda outras características de interesse que, como extensão deste trabalho, possam ser contempladas, como por exemplo, a visualização da evolução comportamental do software ou até mesmo a evolução de suas linhas de código fonte.

A Figura 1 apresenta três elementos chave desta proposta, que serão, ao longo dos

demais capítulos, detalhados amplamente: ***Fonte de Dados***, ***EvolTrack***, ***Visualizadores***. A idéia é que o mecanismo proposto, ***EvolTrack***, seja capaz de coletar informações de projeto de determinadas ***Fonte de Dados*** e, posteriormente, seja capaz de apresentar em sistemas ***Visualizadores*** de maneira evolutiva, utilizando uma dada representação, toda história de evolução estrutural de um determinado projeto de software.

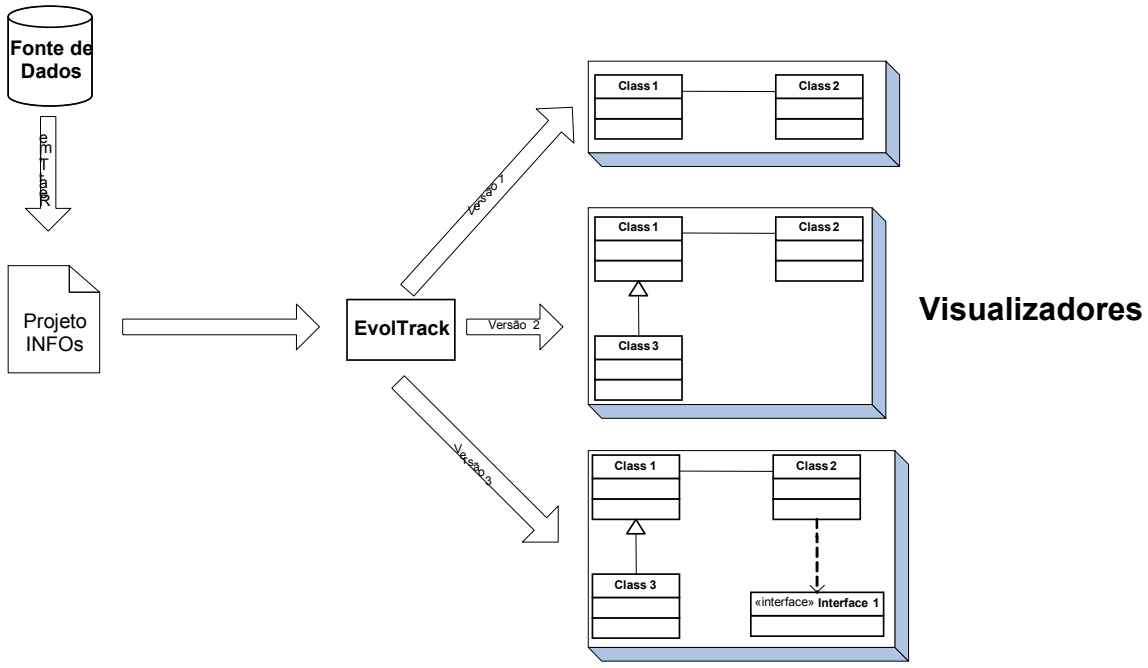

Figura 1. Uma visão de utilização da abordagem proposta.

É importante salientar que por ***Fonte de Dados*** entende-se qualquer sistema externo ao mecanismo que possua armazenado, ou que tenha formas de recuperar, informações históricas e/ou atuais sobre a estrutura de um determinado software em andamento ou já finalizado. Neste contexto, podemos citar como exemplo os sistemas de controle de versão utilizados pelos projetos de software atualmente.

Em contrapartida, sistemas chamados de ***Visualizadores*** serão entendidos por este trabalho como quaisquer sistemas externos ao mecanismo que sejam capazes de apresentar, utilizando alguma forma de representação, as informações de evolução

fornecidas pelo ***EvolTrack***. Exemplos de tais sistemas podem ser ambientes de modelagem de software ou até mesmo, de forma mais geral, ambientes de desenvolvimento de software (HARRISON et al., 2000).

Diversos trabalhos relacionados com a visualização da evolução de software já solucionaram de diferentes maneiras as questões apresentadas anteriormente, onde, a partir de uma fonte de dados, as informações de projeto são extraídas e apresentadas utilizando uma determinada notação. Alguns destes trabalhos serão detalhados no Capítulo 2.

Adicionalmente, o presente trabalho propõe um importante requisito para o objetivo apresentado previamente que, em conjunto com a notação adotada para representar as informações de projeto (detalhada no Capítulo 3), representa uma das principais contribuições desta nova abordagem. Este requisito implica em uma abordagem que seja compatível arquiteturalmente com qualquer fonte de dados e qualquer tipo de visualizador.

Ao contrário dos trabalhos relacionados, que serão apresentados no Capítulo 2, o mecanismo proposto não pressupõe a utilização de qualquer tipo de fonte de dados ou visualizador. Assim, introduz-se a necessidade da abordagem ser facilmente extensível e reutilizável em diversos cenários de utilização.

Por exemplo, uma dada organização possui informações da evolução de seus projetos em dois sistemas de versionamento **Fonte1** e **Fonte2**. Então, em um primeiro momento, esta deseja visualizar a evolução de seus projetos armazenados em **Fonte1** em um dado dispositivo de visualização **Visual1**. Desta forma, a organização deverá instanciar a abordagem informando que a fonte de dados **Fonte1** será utilizada, enquanto que o visualizador **Visual1** será utilizado. Posteriormente, quando se desejar obter as

informações de projeto contidas na **Fonte2**, bastará que a organização informe para o mecanismo que uma nova fonte de dados, **Fonte2**, será utilizada. Note que neste caso, não deverá haver qualquer tipo de esforço por parte da organização usuária quanto a recodificações do mecanismo para fins de viabilizar esta nova integração. Tão pouco, o visualizador deverá apresentar as informações em uma notação diferente da utilizada anteriormente. Apenas o conteúdo apresentado pelo visualizador **Visual1** deverá mudar, em função da troca de fonte de dados.

Entretanto, é importante ressaltar que os recursos necessários para a comunicação com as fontes de dados **Fonte1** e **Fonte2** e com o visualizador **Visual1** deverão estar disponíveis previamente. Isto é, uma interface bem definida de comunicação, entre a abordagem e estes elementos externos, deverá ser especificada para contemplar tal objetivo.

Resumidamente, o objetivo do trabalho pode ser expresso da seguinte forma:

O1. Proporcionar um mecanismo de visualização capaz de representar todo o ciclo de vida estrutural de um projeto de software a partir de informações previamente coletadas sobre o mesmo;

1.3. Organização do texto

Esta monografia está organizada ao longo de 4 capítulos, além deste capítulo de introdução. O segundo capítulo exibe algumas abordagens de visualização da evolução de sistemas de software existentes na literatura. Adicionalmente, uma comparação entre estas abordagens é realizada, a partir de critérios definidos no mesmo capítulo; O terceiro capítulo apresenta os principais requisitos de um mecanismo que deva atender ao objetivo aqui descrito. A partir destes requisitos, um conjunto de problemas associados é

identificado e suas respectivas soluções são apresentadas, caracterizando, assim, uma abordagem para a construção deste mecanismo; No quarto capítulo, uma implementação desta abordagem é discutida e apresentada; No quinto capítulo são delineadas as principais contribuições e limitações deste trabalho, assim como algumas propostas futuras de extensão do trabalho e integração com outras abordagens.

Capítulo 2. Revisão da Literatura

2.1. Introdução

Este capítulo apresenta um conjunto de ferramentas e abordagens para apoiar o processo de compreensão sistêmica durante o ciclo de vida do software. Entretanto, será dado maior enfoque em ferramentas que utilizem visualização como principal recurso no auxílio deste processo. Desta forma, foram analisadas as seguintes ferramentas: GASE (HOLT e PAK, 1996), EvoLens (RATZINGER et al, 2005), CVSscan (VOINEA et al., 2005), softChange (GERMAN et al., 2006) e Augur (FROEHLICH e DOURISH, 2004).

O restante do capítulo está organizado da seguinte forma: A seção 2.2 apresenta as funcionalidades da ferramenta GASE e uma breve discussão sobre sua abordagem; A seção 2.3 apresenta as funcionalidades da ferramenta EvolLens e uma breve discussão sobre sua abordagem; A seção 2.4 apresenta os recursos da ferramenta CVSscan e uma breve discussão sobre sua abordagem; A seção 2.5 apresenta as funcionalidades da ferramenta softChange e uma breve discussão sobre sua abordagem; A seção 2.6 apresenta as funcionalidades da ferramenta 2.6 e uma breve discussão sobre sua abordagem; A seção 2.7 apresenta os recursos da ferramenta Augur e uma breve discussão sobre sua abordagem; A seção 2.8 apresenta uma comparação entre as diferentes abordagens apresentadas no capítulo; A seção 2.9 conclui o capítulo, sumarizando as principais contribuições e limitações das abordagens discutidas.

2.2. GASE

A ferramenta GASE recebe descrições de sucessivas versões da arquitetura de um determinado software e, a partir destas, gera um conjunto de diagramas que reflete esta

evolução. O diagrama gerado é comparativo, ou seja, sempre compara duas versões V_{n-1} e V_n do sistema.

Arquiteturalmente, a ferramenta é composta por quatro partes funcionais: *extrator*, *analisador*, *mapeador*, e um *visualizador*. O extrator tem por objetivo percorrer todo código legado da aplicação alvo, cujo código fonte encontra-se escrito em linguagem C (KERNIGHAN e RITCHIE, 1978), determinando que artefatos serão relevantes para a análise posterior. Estes artefatos de relevância arquitetural são determinados e descritos através de uma notação própria, conforme descrito por HOLT e MANCORIDIS (1994). A partir dos artefatos gerados pelo extrator em duas versões, o analisador realiza uma separação, marcando os artefatos exclusivos da primeira versão, comuns em ambas as versões e exclusivos da segunda versão.

Então, o *mapeador* transforma estas marcações em diagramas com nós (ex.: módulos) e arestas (ex.: dependências) coloridos, de acordo com as alterações realizadas da versão mais antiga para a mais recente. A seguinte regra de coloração é adotada na visualização:

- Vermelho para artefato adicionado;
- Cinza para artefato comum entre ambas as versões;
- Azul para artefato removido.

A Figura 2 exemplifica este esquema de coloração. Note que, neste diagrama fictício, temos dois módulos adicionados (módulo 4 em vermelho, módulo 3 em vermelho), um inalterado (módulo 2 em cinza) e um removido (módulo 1 em azul).

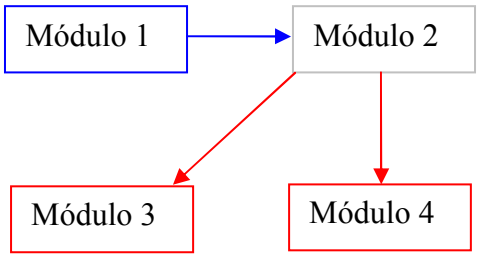

Figura 2. Exemplo de diagrama gerado pelo GASE.

Adicionalmente, existe o elemento *visualizador*, que representa um ambiente de apresentação dos diagramas gerados pelo *mapeador*. Através deste ambiente, são disponibilizadas funcionalidades como *zoom*, mover a câmera na mesma direção do objeto (i.e. *panning*), navegação e consulta a artefatos.

Desta forma, esta ferramenta tem como principais contribuições, a identificação automática de elementos estruturais do software e sua posterior apresentação visual. Possui um bom conjunto de recursos de interação com o diagrama, proporcionando uma boa experiência ao usuário da mesma. Entretanto, pode ser observado que a necessidade de interação com usuário para obtenção das informações de um determinado sistema de software representa uma importante limitação da ferramenta. Além disto, a utilização de uma notação própria e pouco difundida pode, de certa forma, aumentar o grau de complexidade no aprendizado da ferramenta, uma vez que o usuário deverá, *a priori*, entender detalhadamente os elementos desta nova notação.

2.3. EvoLens

A iniciativa chamada EvoLens é uma abordagem de visualização para acompanhar a evolução de um determinado software a partir de múltiplas dimensões. Esta abordagem foi desenvolvida para atuar sobre sistemas orientados a objetos. Como

fonte de informação para os dados do software, a ferramenta foi projetada para utilizar sistemas de versionamento. Especificamente falando, a ferramenta suporta dados de software desenvolvido em linguagem Java (SUN MICROSYSTEMS, 2008) e mantidos dentro de um sistema de versionamento CVS (CEDERQVIST et al., 2006).

O tempo encontra-se discretizado pelas datas geradas pelo sistema de versionamento. Isto é, a evolução se dá a partir de *check-in* de código no CVS, enquanto que a hierarquia do software é representada por sua própria estrutura. Esta atividade de *check-in* pode ser entendida como uma nova revisão para um determinado arquivo, em geral, contemplando um conjunto de modificações no mesmo (ESTUBLIER, 2000).

Neste contexto, os mecanismos de visualização desta estrutura utilizam grafos como forma de representação e cores são utilizadas para representar mudanças nas partes do software. Adicionalmente, um modelo de visualização tipo lente (SARKAR e BROWN, 1992) (*lens-view*) é utilizado com o intuito de reduzir a quantidade de informação apresentada ao usuário da ferramenta.

Desta forma, o engenheiro de software (usuário da ferramenta), doravante denominado engenheiro, é capaz de selecionar um ponto focal na visualização que define o centro da representação feita pela ferramenta. Assim, apenas os módulos ou classes dentro deste ponto focal e relacionamentos que iniciem neste serão apresentados.

A Figura 3 apresenta uma pequena amostra desta técnica. Neste caso, o ponto focal foi configurado para o módulo *jvision*. Este contém cinco sub-módulos, a saber: *overlay*, *image*, *vis*, *applet* e *main*.

Cada nó no grafo representa uma classe e cada aresta representa uma relação de acoplamento de mudança entre duas classes. Isto é, se a alteração de uma determinada classe é acompanhada pela alteração de outra classe, estas terão uma relação de

acoplamento de mudança (uma aresta no grafo). Quanto maior a freqüência que uma determinada classe é alterada em conjunto com uma outra classe, maior será a espessura da aresta.

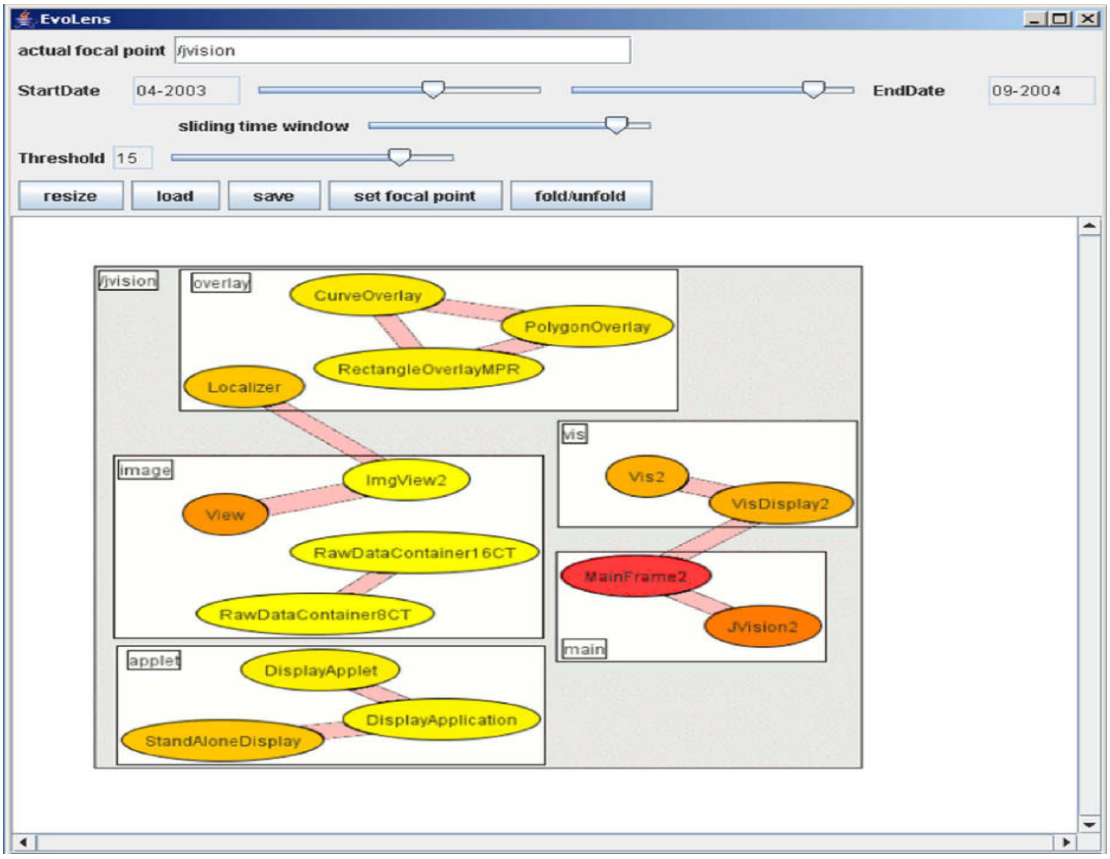

Figura 3. A ferramenta EvoLens com ponto focal no módulo jvision (Fonte: RATZINGER et al. (2005)).

Como podem ser observadas na Figura 3, cores são utilizadas para indicar métricas de crescimento. As cores (, , , e) representam cinco níveis de crescimento diferentes das classes. Começando com o amarelo claro, baixo nível de crescimento, e terminando com o vermelho escuro, alto índice de crescimento. Neste contexto, o conceito de crescimento está atrelado à quantidade de linhas de código alteradas ao longo do tempo. Estas informações, assim como todas as outras utilizadas pela ferramenta, são originadas a partir de dados históricos extraídos do sistema de

controle de versão CVS.

Este cálculo de crescimento e de acoplamento é feito dentro de uma janela de tempo especificada na ferramenta. Assim, é importante ressaltar que o conceito de evolução nesta ferramenta está atrelado a métricas como índice de crescimento e índice de acoplamento entre classes, ao contrário da evolução estrutural do software.

Portanto, esta ferramenta apresenta como principal contribuição uma forma de visualização ao longo do tempo de atributos como nível de crescimento e de acoplamento entre diferentes classes Java de um determinado sistema de software. Utiliza como fonte de informação o amplamente difundido e utilizado sistema de controle de versão CVS. Além disto, possui recursos de interação e manipulação do diagrama apresentado como efeitos de *zoom* e filtragem de elementos para serem exibidos. Porém, a ferramenta não proporciona qualquer tipo de mecanismo que possibilite a utilização de uma outra fonte de dados específica, como, por exemplo, o sistema de controle de versão Subversion (SUSSMAN et al., 2004). Tão pouco possibilita a utilização de uma notação alternativa para a visualização das informações providas.

2.4. CVSscan

O CVSscan é uma ferramenta de apoio ao processo de manutenção e entendimento de software que, através da técnica de visualização, expõe ao engenheiro diversos aspectos, ao longo do tempo, de uma determinada aplicação sob análise. Entretanto, ao contrário das demais ferramentas citadas anteriormente, o CVSscan utiliza uma abordagem baseada em linhas de código para a visualização do software.

Nesta abordagem (*pixel line*), pontos na tela são utilizados para representar, sob algum tipo de perspectiva, linhas de código fonte. Cores distinguem as possíveis

variações de uma perspectiva. Por exemplo, na ferramenta CVSscan existem basicamente três perspectivas, ou dimensões, onde as linhas de código são classificadas: *status* da linha, tipo de construção e autor. A perspectiva de *status* da linha classifica-a em uma das seguintes categorias: constante (i.e. linha inalterada em relação à versão anterior), a ser inserida, modificada e removida. A perspectiva de tipo de construção classifica funcionalmente a linha de acordo com a linguagem de programação utilizada. Por exemplo, se a linha for um comentário no programa, será classificada com uma categoria de mesmo nome. Já a perspectiva de autor, classifica a linha de acordo com o autor da linha. Cada autor que realiza alterações no software terá uma cor diferente. A Figura 4 ilustra algumas colorações utilizadas para cada perspectiva.

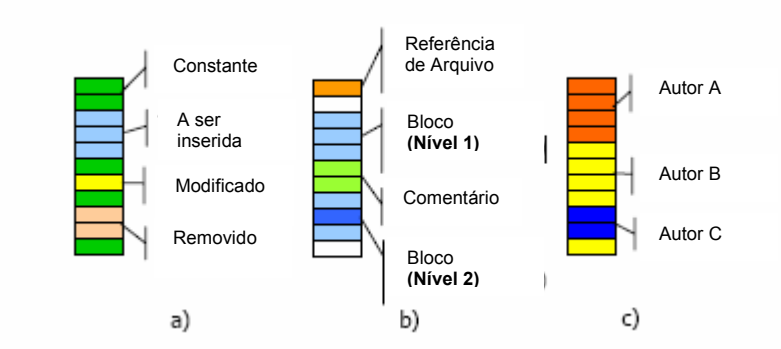

Figura 4. Colorações utilizadas: status da linha (a), tipo de construção (b) e autor (c).

Todas estas linhas de código utilizadas, que representam a principal fonte de informação da aplicação, são originadas a partir de sistemas de controle de versão. Atualmente, apenas o sistema CVS é suportado pela ferramenta. Nesta implementação, uma outra ferramenta, chamada CVSgrab, é responsável por extrair as informações do CVS e repassar para o CVSscan para o devido processamento. Desta forma, podemos reparar que houve uma tentativa de se criar uma arquitetura modular para que no futuro outros sistemas de controle de versão pudessem ser utilizados com o CVSscan. Além

disto, pôde ser observado que a ferramenta suporta a análise tanto de arquivos fonte escritos em linguagem C como arquivos fonte escritos em linguagem Perl (WALL et al., 2000).

Entretanto, pela simples análise da ferramenta, não ficou claro em que tipo de formato estes dados trafegam entre o CVS e o CVSgrab (VOINEA e TELEA, 2006) e, posteriormente, entre o CVSgrab e o CVSscan, impossibilitando assim inferir o quão independente a ferramenta CVSscan é em relação ao sistema CVS.

Neste contexto, cada ponto discreto (i.e. versão) no ciclo de vida de um arquivo é representado pela ferramenta a partir da tupla: (identificação da versão, autor, data, código). Então, para comparar versões consecutivas de um determinado arquivo a ferramenta utiliza uma aplicação externa nos moldes de um *diff* (HUNT e MCILROY, 1976) (HUNT e SZYMANSKI, 1977) do sistema operacional UNIX. Assim, a partir da saída desta aplicação, o CVSscan rotula cada linha de acordo com as categorias de *status* de linha citadas anteriormente.

A

Figura 5 ilustra como ocorre o processo de visualização na ferramenta. É interessante notar que a ferramenta não utiliza indentação e tamanho da linha para representar uma possível estrutura para o arquivo. Ao contrário, utiliza-se de um tamanho fixo de linhas, maior ou igual ao maior número de linhas atingido pelo arquivo ao longo do tempo, e cores para codificar a estrutura.

Cada linha vertical representa uma versão do arquivo e cada linha horizontal representa uma linha do arquivo, conforme pode ser observado na parte central da ferramenta (marcada como "*Evolution Overview"*). Adicionalmente, podem ser observadas, nas bordas, métricas que complementam a visualização. Na borda esquerda,

uma barra vertical representa o tamanho da versão em linhas, codificada através de cores. Na borda inferior, uma barra horizontal é utilizada para representar o autor responsável por cada versão.

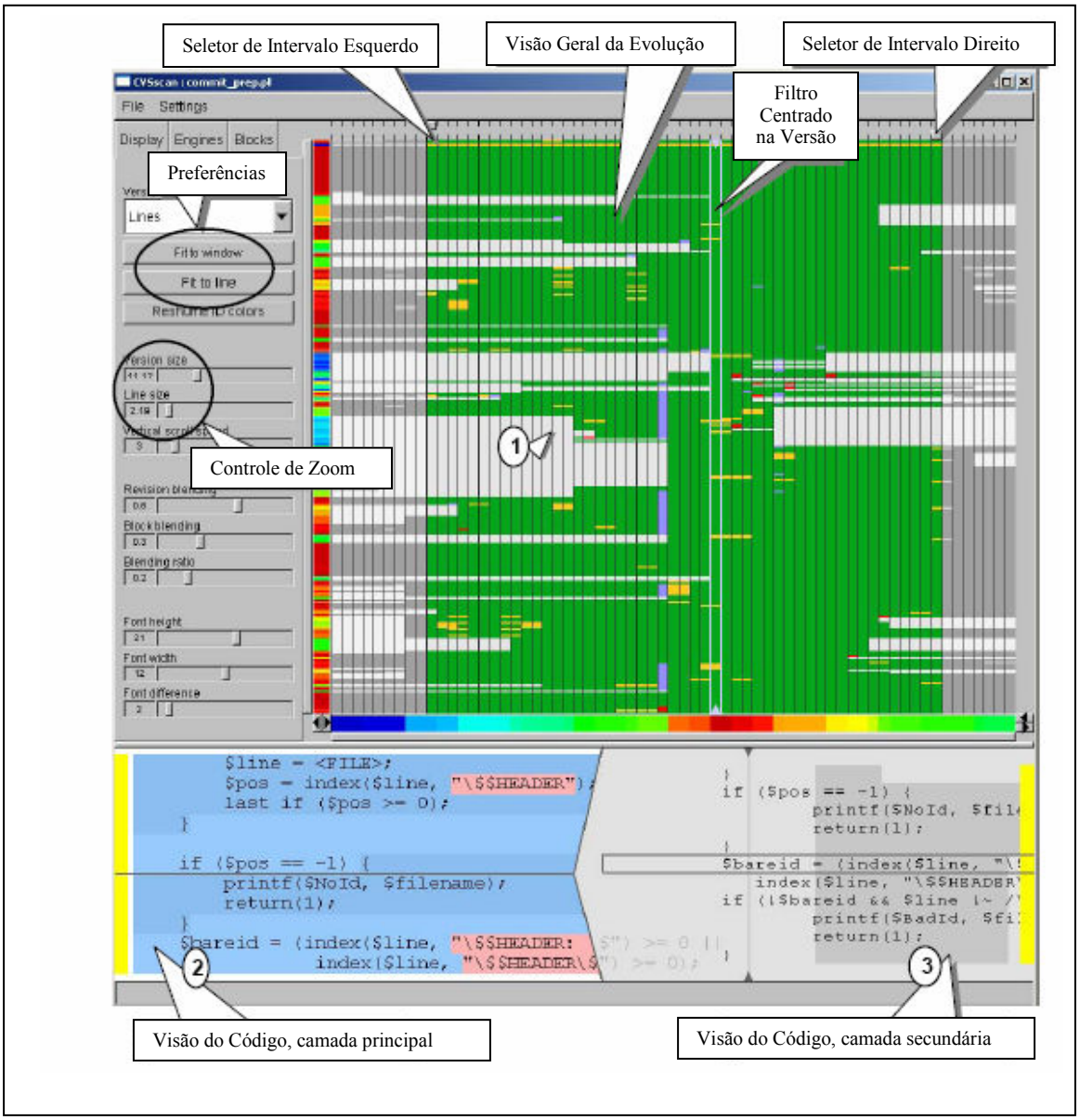

Figura 5. Visão geral da ferramenta CVSscan (Fonte: VOINEA et al. (2005)) .

Uma outra funcionalidade interessante está sendo exemplificada na

Figura 5. Note que ao passar o *mouse* sobre uma parte da visualização (marcado como (1) na figura), ocorre respectivamente uma apresentação do código referente a esta

passagem (marcado como (2) e (3) na figura). Além disto, a ferramenta oferece alguns recursos adicionais como *zoom*, alteração do tamanho das fontes exibidas, alteração do tamanho das linhas exibidas, seleção dos intervalos de tempo para exibição (através do "*Seletor de Intervalo Esquerdo"* e do "*Seletor de Intervalo Direito"*) e outros.

É importante ressaltar que toda análise realizada pela ferramenta, e conseqüentemente todo resultado gerado por esta, tem como único foco arquivos e suas respectivas linhas. Isto é, a ferramenta é capaz apenas de representar, ao longo do tempo, a evolução de um único arquivo por vez, ou seja, representar como suas linhas foram sendo acrescidas, removidas ou alteradas ao longo da execução do projeto.

2.5. softChange

Tomando como ponto de partida as limitações estabelecidas pelo sistema de controle de versão CVS, no que tange ao entendimento da evolução de seus projetos sob controle, criou-se a ferramenta softChange. Esta tem por objetivo aproveitar informações disponíveis de projeto e, a partir destas, apresentar diferentes características do software ao longo do tempo.

A abordagem da ferramenta define como *rastros de software* informações deixadas pelos contribuidores de um determinado projeto ao longo do processo de desenvolvimento, como listas de e-mail, *web sites*, registros do sistema de controle de versão, *releases* de software, documentação e código fonte, dentre outros. Então, a idéia da ferramenta é, a partir de repositórios CVS, transformar estes *rastros de software*, utilizando determinadas heurísticas, em informações de mais alto nível que serão posteriormente apresentadas para o usuário final. Por exemplo, posteriormente, poderia ser apresentado, ao longo da linha do tempo, um reagrupamento do conjunto de revisões dos arquivos do projeto para um dado evento de *check-in* no repositório. Isto se torna útil

neste cenário, uma vez que o sistema CVS não é orientado a transação, ou seja, o mesmo não mantém qualquer tipo de rastro ou ligação entre os diferentes arquivos modificados em um dado *check-in*.

A partir deste processo de extração dos *rastros de software* e da reconstrução de todos os eventos de *check-in* do projeto, uma etapa de análise é realizada. Nesta, classifica-se o evento como sendo uma adição de nova funcionalidade, uma reorganização de código fonte, um simples comentário, e etc. Por fim, após estas atividades de extração e análise, o softChange provê uma representação gráfica destas informações. A ferramenta também disponibiliza uma representação destas informações utilizando documentos contendo hipertextos para serem visualizados em navegadores WEB.

A arquitetura da ferramenta pode ser observada na Figura 6. Nesta, nota-se a existência de quatro componentes básicos: repositório de *rastros de software*, extrator de *rastros de software*, analisador de *rastros de software* e um visualizador.

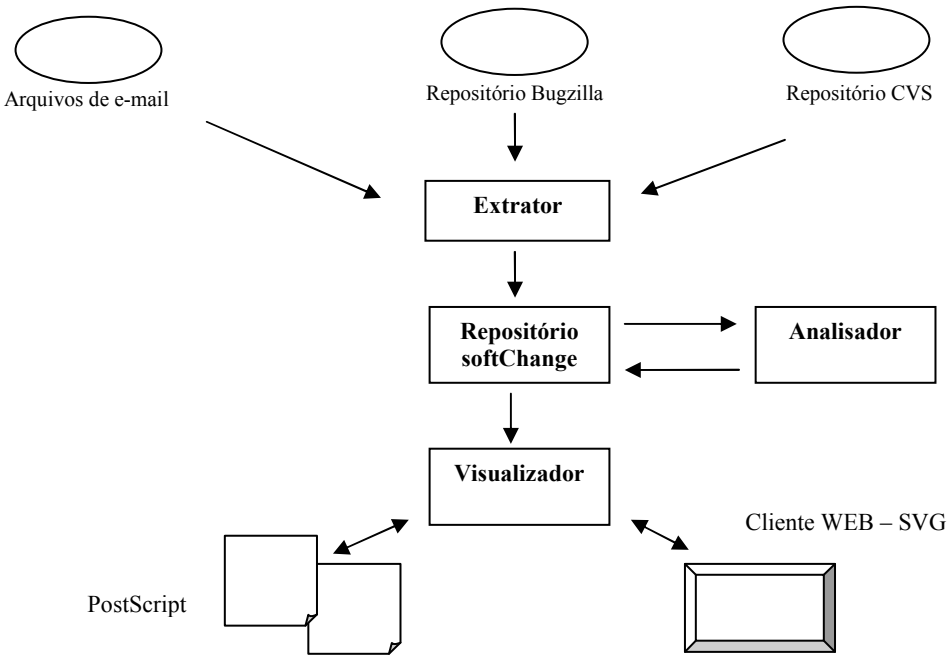

Figura 6. Arquitetura da ferramenta softChange.

O repositório de *rastros de software* é um banco de dados relacional responsável por armazenar todas as informações históricas extraídas sobre o projeto. O extrator de *rastros de software* é o componente responsável por resgatar as informações de projeto das fontes de dados suportadas. Nesta implementação, as fontes de dados suportadas são: o repositório CVS, arquivos do tipo *ChangeLog*, *releases* do software (através de arquivos compactados distribuídos pelos membros da equipe) e o sistema Bugzilla (BUGZILLA, 2008). O componente analisador de *rastros de software* é responsável por aplicar as heurísticas definidas pela abordagem e, a partir das informações previamente coletadas, inferir novas informações e características de software. Por exemplo, este componente possui a inteligência de correlacionar eventos de *check-in* realizados ao longo do tempo com problemas cadastrados na ferramenta Bugzilla.

O último componente é o visualizador. Este é divido em duas partes: um interpretador de hipertexto (i.e. *browser*) e um visualizador gráfico. O primeiro é utilizado pelo usuário para navegar através dos diversos *check-ins* realizados no projeto ao longo do tempo. Esta navegação pode ser realizada por data, autor ou arquivo. Em uma navegação por data, por exemplo, o softChanges provê informações de que revisões dos arquivos pertencem a cada *check-in*, além de outros metadados associados à modificação.

O visualizador gráfico também pode ser subdividido em duas partes principais. A primeira utiliza arquivos Postscript para gerar gráficos estáticos dos *rastros de software*. Um exemplo de gráfico gerado em Postscript é ilustrado na Figura 7 (neste caso, o autor chama de MR o que está sendo chamado de *check-in* neste texto). A segunda utiliza os recursos de SVG (SVG, 2008) para apresentar tais informações de forma interativa.

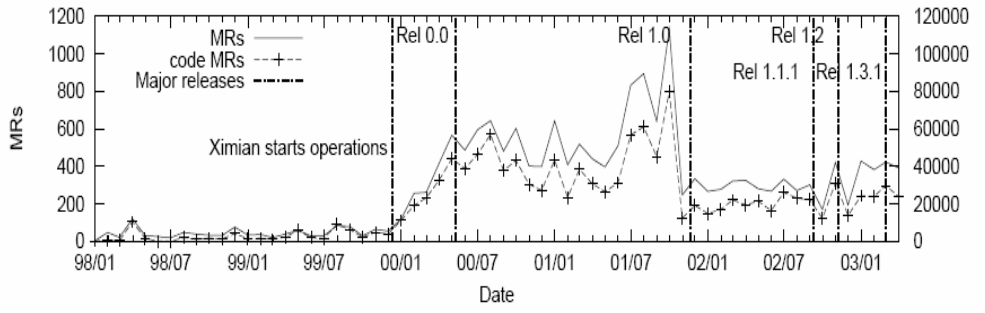

Figura 7. Exemplo de um gráfico gerado pelo softChange para o projeto *open source* Ximian (Fonte: GERMAN et al. (2006)).

A ferramenta softChange apresenta como principais contribuições a integração com mais de uma fonte de dados para a coleta de informações acerca do projeto, uma arquitetura modular onde papéis e responsabilidades foram bem definidos, aumentando a facilidade de sua extensão. Além disto, proporciona um método para a visualização gráfica de diferentes propriedades de um dado projeto de software, como: crescimento de linhas de código ao longo do tempo, número de *check-ins* realizados ao longo do tempo, número de arquivos criados ao longo do tempo e número de arquivos por evento de *check-in*, dentre outros.

Porém, sua principal limitação aparenta ser seu forte vínculo temporal com sistemas de controle de versão. Isto é, por construção, a ferramenta pode apenas apresentar a evolução dada discretamente no tempo, onde cada ponto no espaço temporal representa um evento de *check-in* em um dado repositório de versionamento de código.

2.6. Augur

A ferramenta Augur é um sistema de visualização que tem como objetivo prover informações, ao longo do tempo, da estrutura de artefatos de um projeto e de suas atividades relacionadas. Isto é, a ferramenta parte da premissa de que, em um processo de

desenvolvimento de software, existem basicamente dois tipos de complexidade que os desenvolvedores costumam lidar. A primeira é inerente aos artefatos gerados ao longo do processo, neste caso, trata-se de código fonte. A segunda complexidade refere-se às atividades por onde tais artefatos são gerados. Desta forma, esta abordagem procura, além de apresentar a evolução dos elementos de um projeto ao longo tempo, correlacionar tais eventos geradores de mudança às atividades estabelecidas ao longo do projeto.

A ferramenta utiliza uma abordagem de visualização baseada em linhas de código fonte (EICK et al., 1992). Desta forma, a ferramenta associa a cada linha de código um conjunto de informações de interesse, como a que atividade esta se refere, a que método ou classe esta se refere e quem criou a linha, dentre outras informações. Fazendo com que estas carreguem em si todas as informações necessárias para a sua compreensão e para a sua correlação com determinadas atividades do projeto.

Adicionalmente, é interessante observar que a ferramenta possui um conjunto muito bem definido de decisões arquiteturais. Por exemplo, a ferramenta foi desenvolvida para trabalhar de modo concorrente ao desenvolvimento do projeto, ao contrário de trabalhar de forma retrospectiva, isto é, baseado em dados históricos do projeto. Em outras palavras, as informações providas pela ferramenta são atualizadas em tempo de desenvolvimento. Enfatizando o requisito imposto pela proposta de não apenas servir como um meio pelo qual desenvolvedores possam entender um determinado projeto, mas também ser útil nas atividades de coordenação de equipes geograficamente distribuídas.

Além disto, a ferramenta tem como requisito alta interoperabilidade. Desta forma, foi implementada uma arquitetura que permita a utilização de diferentes tipos de sistemas de versionamento como fonte de dados, assim como diferentes tipos de analisadores de

código.

No que tange a questão da visualização dos dados, a ferramenta apresenta em uma única tela, porém em painéis distintos, diferentes características do projeto em andamento. Sua parte central, e principal, disponibiliza ao usuário uma visão temporal da evolução das linhas de código fonte do projeto. Níveis de coloração são estabelecidos para identificar o quão recente determinada linha foi modificada. Note que ao selecionar uma linha neste painel, informações associadas a mesma são exibidas nos demais. Por exemplo, no painel esquerdo são exibidos dois tipos de informação: informação do autor que modificou a linha no instante de tempo selecionado e o tipo de estrutura à qual esta linha está associada (p.ex. método, classe, comentário). O painel inferior contém informações adicionais como, informações de *check-in* e outros metadados do sistema de controle de versão. A

Figura 8 apresenta a tela da ferramenta e suas funcionalidades.

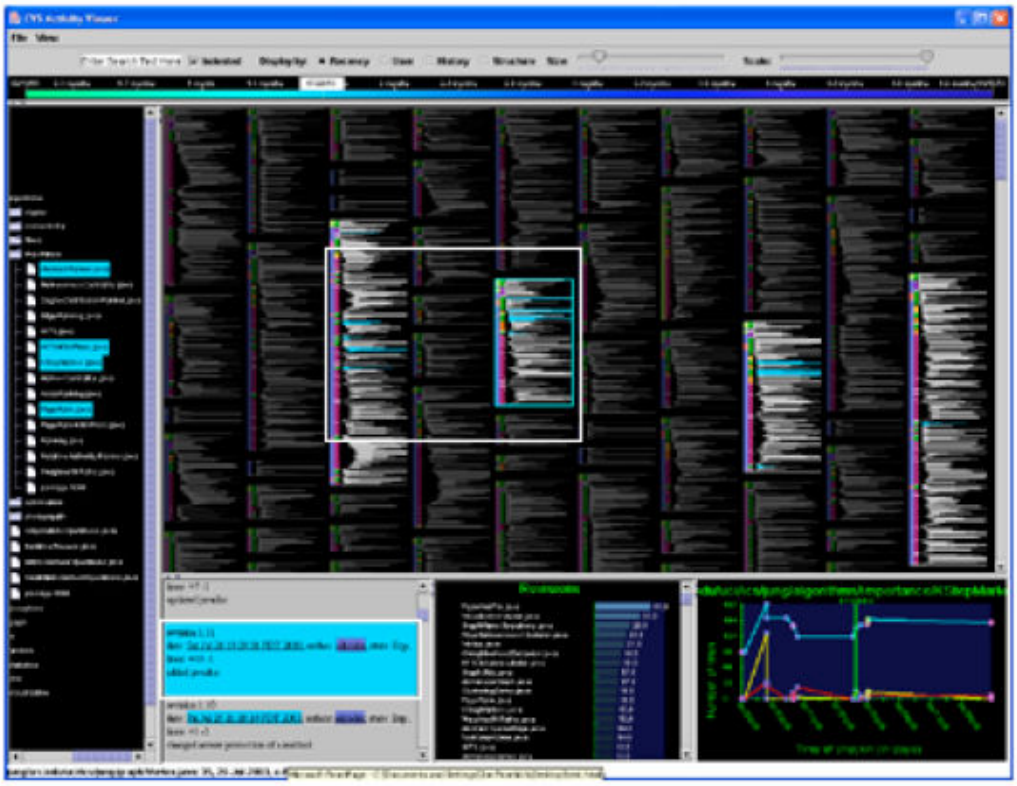

Figura 8. Tela da ferramenta Augur (Fonte: FROEHLICH e DOURISH (2004)).

O Augur utiliza como representação interna para as informações extraídas do projeto uma estrutura de dados baseada em linhas de código. Esta por sua vez é incrementada, posteriormente, com informações adicionais como informações de indentação, data, autor, revisão, um nó de uma árvore sintática abstrata (AHO et al., 2006) e informações da estrutura à qual esta linha pertence.

Como a ferramenta utiliza para a tarefa de análise de código fonte o sistema ANTLR, um gerador de *parser* baseado na tecnologia Java, esta pode suportar, em teoria, diversos tipos de linguagens de programação. Entretanto, no contexto desta implementação, foi adotada apenas a análise de código fonte escrito em Java.

Portanto, a ferramenta Augur representa uma outra abordagem para tratar do problema da visualização de software ao longo do tempo. A ferramenta utiliza para apresentação uma abordagem baseada em linhas de código fonte. Associado a estas, também são apresentados outros tipos de informação, como a sua estrutura de mais alto nível associada (como método, classe ou pacote) e o autor das diferentes modificações que ocorreram ao longo do tempo. Ao contrário das demais ferramentas, esta apresenta uma solução arquitetural que possibilita sua utilização com diferentes tipos de fontes de dados e analisadores de linguagem, aumentando, desta forma, sua capacidade de utilização.

2.7. Outras abordagens

Além das ferramentas apresentadas anteriormente, pode ser citada também a ferramenta GEVOL (COLLBERG et al., 2003). Esta também representa um sistema de

visualização da evolução de software que utiliza como única fonte de dados o sistema de controle de versão CVS. A ferramenta é compatível apenas com a linguagem de programação Java. Sua principal contribuição é a forma graficamente chamativa de representação visual (GAJER e KOBOUROV, 2000) das informações do projeto. A Figura 9 apresenta um cenário de utilização da ferramenta, onde diferentes elementos do software são expostos.

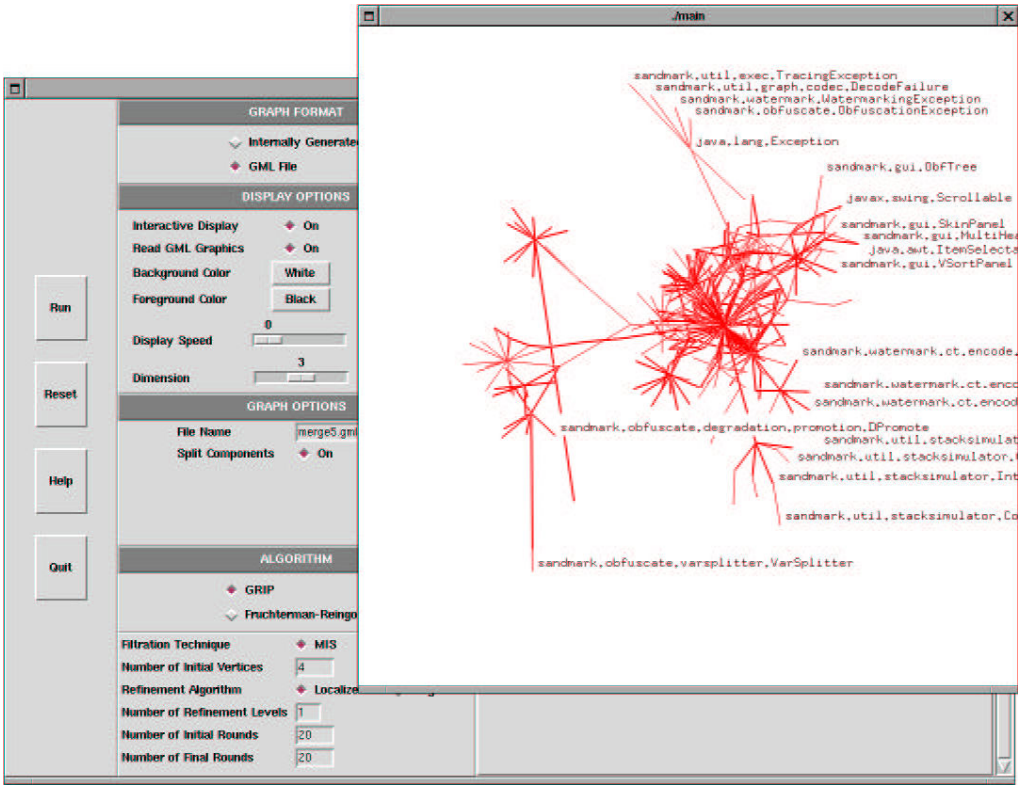

Figura 9. A ferramenta GEVOL sendo utilizada.

Outros trabalhos (LANZA, 2001), (WU, et al., 2004) e (WU, 2003) não detalhados neste texto, pois possuem exatamente o mesmo conjunto de características dos trabalhos apresentados, também abordam o tema de visualização da evolução do software sob determinadas perspectivas e apresentando variadas características de interesse.

2.8. Comparação entre as ferramentas

Os critérios abaixo descritos foram definidos de acordo com as características comuns encontradas em todas as ferramentas pesquisadas e de acordo com o conjunto de características apresentadas no Capítulo 1 para uma abordagem reutilizável e extensível para o problema de visualização da evolução de sistema de software. Desta forma, as ferramentas descritas neste capítulo foram comparadas de acordo com os seguintes critérios:

C1. Tipo de fonte dados utilizada: Indica que outra ferramenta externa armazena os dados utilizados sobre o sistema em análise.

C2. Tipo de visualização utilizada: Indica que tipo de recurso visual foi adotado para representar as características do sistema em análise.

C3. Granularidade das informações apresentadas: Indica o nível de granularidade das informações extraídas do sistema em análise que são apresentadas.

C4. Integração com algum ambiente de desenvolvimento de software: Indica se a ferramenta possui algum tipo de integração com ambientes de desenvolvimento de software, podendo extrair, alterar ou apresentar informações destes ambientes.

C5. Atualização visual automática: Indica se a representação visual apresentada ao usuário sofre atualização automática, conforme novos dados são gerados e extraídos da fonte dados, ou se é preciso manualmente atualizar a representação.

C6. Análise Temporal: Indica se a abordagem em questão proporciona ao seu usuário informações da evolução do sistema sob análise em espaços discretos

do tempo, continuamente no tempo, ou ambas as estratégias.

A Tabela 1 classifica e caracteriza as ferramentas de acordo com os critérios enunciados. A classificação dos dois primeiros critérios corresponderá a uma descrição por extenso do nome de sistemas, tecnologias ou metodologias utilizadas, que atendam ao critério em questão. A classificação do terceiro critério será dada como Alta ou Baixa. Granularidade alta será associada a informações de alto nível do sistema em análise, como módulos, pacotes, classes e subsistemas. Granularidade baixa será associado a informações de menor grão do sistema em análise, como métodos, atributos ou até mesmo linhas de código fonte sem qualquer tipo de classificação. Adicionalmente, os dois últimos critérios serão respondidos da seguinte forma: ✓ para verdadeiro e ✗ para falso.

	Ferramentas				
Critério	**GASE**	**EvoLens**	**CVSscan**	**softChange**	**Augur**
C1	Usuário	CVS	CVS	CVS	Variável
C2	Diagrama próprio	Diagrama próprio + Lens-View	Gráfico Baseado em Linhas	Gráfico Baseado em *Check-in*	Gráfico Baseado em Linhas
C3	Alta	Alta	Baixa	Baixa	Variável
C4	✗	✗	✗	✗	✗
C5	✗	✗	✗	✗	✓
C6	Ambas	Discreta	Discreta	Discreta	Discreta

Tabela 1. Quadro comparativo entre as ferramentas analisadas.

A partir desta comparação, concluímos que as ferramentas pesquisadas, de um modo geral, estão vinculadas a determinados sistemas de controle de versão para captar as informações do sistema que deverá ser analisado. Assim, em função da constante evolução tecnológica e de possíveis mudanças no ferramental utilizado na gerência de

configuração, isto se torna um grande agravante quando pensamos em abordagens que perdurem ao longo do tempo com certa estabilidade. Desta forma, a criação de uma abordagem alternativa que possibilite a utilização de outros tipos de fonte de dados, com pouco ou até mesmo sem nenhum impacto na arquitetura da ferramenta se faz necessária.

Adicionalmente, vemos que em nenhum caso um ambiente de desenvolvimento de software foi sequer mencionado como uma possível fonte de dados, limitando os usuários das ferramentas a uma análise de evolução de software discreta no tempo, onde cada *check-in* no sistema de controle versão representa uma "foto" do sistema no tempo. Assim, também, podemos citar como um fator motivador a criação de uma abordagem que possibilite a mesma análise da evolução, porém, continuamente no tempo. Isto é, com uma integração entre a ferramenta e o ambiente de desenvolvimento de software, temos como rastrear cada passo do desenvolvedor e, assim, gerar evidências de uma evolução contínua.

2.9. Conclusão

Neste capítulo foram apresentadas algumas abordagens, concretizadas na forma de ferramentas, que, de certa forma, tratam dos problemas de como apresentar e representar a evolução de um determinado projeto de desenvolvimento de software. É interessante notar que o mesmo problema foi solucionado sob diferentes perspectivas. Por exemplo, em alguns casos focou-se na apresentação da evolução estrutural de um determinado sistema, isto é, a abordagem tinha como objetivo apresentar ao seu usuário como os diferentes elementos que compõem a estrutura do software evoluíram ao longo do tempo. Em outros casos, o mesmo problema foi abordado na granularidade de código fonte, ou seja, a evolução dos arquivos que compõem o software era foco neste caso.

Então, dado um conjunto de características previamente identificadas, as abordagens pesquisadas foram comparadas. Realçando, desta forma, que nenhuma destas atendem a todas as características julgadas necessárias para uma abordagem que trate dos mesmos problemas, porém, de forma reutilizável e extensível.

Os próximos capítulos terão como objetivo apresentar uma abordagem que tem por objetivo atender a estas características de reutilização e extensibilidade identificadas previamente. A esta abordagem dá-se o nome de EvolTrack, conforme será detalhado no Capítulo 3. Adicionalmente, uma implementação para esta abordagem será discutida e apresentada no Capítulo 4.

Capítulo 3. Abordagem Proposta: EvolTrack

3.1. Introdução

Neste capítulo, são apresentados os requisitos de uma abordagem que atenda as necessidades anteriormente discutidas. Adicionalmente, uma descrição de alto nível será apresentada sobre a abordagem proposta, incluindo um diagrama de componentes UML e um modelo conceitual, contendo os principais elementos da abordagem.

Em face do que foi apresentado nos capítulos anteriores, um determinado conjunto de requisitos funcionais, representando características desejadas da abordagem, foi elaborado de forma a contemplar os objetivos deste trabalho.. A lista a seguir enumera estes requisitos:

R1. Permitir a visualização estrutural[1] de um determinado sistema de software sob análise;

R2. Permitir a visualização estrutural ao longo de toda história de desenvolvimento do sistema de software sob análise;

R3. Apresentar uma animação visual da história estrutural do sistema de software sob análise;

R4. Suportar arquiteturalmente a integração com diferentes fontes de dados;

R5. Suportar arquiteturalmente a integração com diferentes tipos de representação da informação;

R6. Ser compatível com o ambiente de desenvolvimento de software Eclipse (ECLIPSE, 2008).

Note que o requisito R2 não restringe o intervalo de tempo utilizado entre cada "foto" da história de um projeto. Isto é, como será visto posteriormente neste capítulo, o

intervalo de tempo que será apresentado na representação do projeto dependerá de como a fonte de dados armazena as informações do projeto e de como o processo responsável por coletar tais informações realizará esta tarefa. Assim, ao contrário das abordagens analisadas no Capítulo 2, é possível notar que a abordagem proposta deve ter a capacidade de representar todas as informações geradas por um projeto ao longo tempo e não apenas os momentos onde ocorreram *check-ins* nos sistemas de controle de versão utilizados.

Além disto, a abordagem proposta deve contemplar arquiteturalmente (vide requisito R4) a utilização de qualquer tipo de fonte de dados que possa fornecer dados referentes aos projetos em análise, não ficando apenas restrita a sistemas de controle de versão, o que aumenta a flexibilidade da abordagem e o potencial de reutilização em outros cenários, isto é, a partir da utilização de outros sistemas de fonte de dados e de visualização.

É também importante notar a integração nativa com o ambiente de desenvolvimento de software Eclipse. Com ampla utilização dentro da comunidade de software, este ambiente está presente em uma grande quantidade de projetos de desenvolvimento de software. Assim, tornamos esta abordagem passível de ser utilizada por qualquer desenvolvedor que tenha algum conhecimento na plataforma Eclipse ou ferramentas similares. Desta forma, ganha-se compatibilidade com a maioria dos projetos de software orientados a objetos desenvolvidos atualmente.

O restante do capítulo apresenta em maiores detalhes a abordagem proposta. A seção 3.2 apresenta uma discussão preliminar sobre a abordagem, ilustrando seus principais elementos; A seção 3.3 apresenta detalhadamente os desafios que deverão ser

[1] Entende-se por visualização estrutural a visualização do conjunto de classes do sistema.

transpostos e suas respectivas soluções, constituindo assim uma abordagem para os requisitos supracitados; A seção 3.4 apresenta um modelo conceitual para a abordagem, servindo de ponto de partida para a implementação discutida no Capítulo 4; A seção 3.5 discute os principais pontos apresentados no capítulo, enfatizando as principais contribuições da abordagem.

3.2. Visão Geral

Como podem ser observados na

Figura 10, dois elementos externos interagem com a abordagem proposta. O primeiro elemento, chamado de Fonte de Dados, é responsável por prover informações sobre o histórico do projeto ao qual o sistema de software sob análise pertence. Exemplos destas fontes de dados podem ser sistemas de controle de versão, como Subversion, CVS, Odyssey-VCS 2 (MURTA et al., 2008), sistema Lighthouse (DA SILVA et al., 2006), e qualquer outro tipo de sistema que tenha acesso a informações de projeto pertinentes.

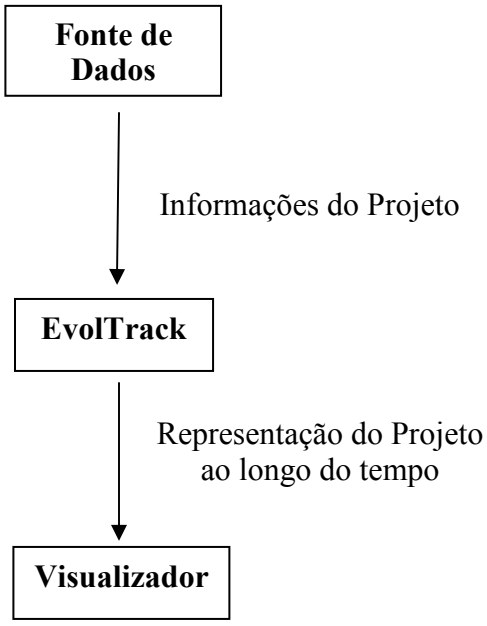

Figura 10. Principais elementos participantes da abordagem.

O segundo elemento é o Visualizador. Este é responsável por apresentar as informações de projeto, ao longo do tempo, de acordo com a representação fornecida pela abordagem. Esta será responsável por traduzir as informações fornecidas pelas fontes de dados em um formato interno padrão. Posteriormente, dada uma política de representação bem definida, as informações descritas neste formato interno serão novamente transformadas em uma representação de saída. Por exemplo, uma representação de saída poderia ter a forma a um diagrama de classe UML (OMG, 2008) ou, alternativamente, poderia apresentar a informação na forma textual para o usuário.

Assim, a abordagem EvolTrack poderia ser composta por um único elemento de alto nível, chamado Núcleo. Como o nome indica, este seria o componente central desta arquitetura. Seu principal objetivo seria gerenciar as informações de projeto extraídas da Fonte de Dados, mantendo toda sua rastreabilidade e orquestrando todo fluxo de informação a ser apresentado. Isto é, caberia ao componente Núcleo a responsabilidade por manter em uma estrutura apropriada todas as versões do projeto geradas ao longo do tempo.

Entretanto, com o intuito de não sobrecarregar o componente Núcleo com todas as responsabilidades funcionais da abordagem, isto é, resgatar, processar e apresentar os dados, uma abordagem modular (SULLIVAN et al., 2001) e baseada em componentes (CRNKOVIC et al., 2004) é proposta.

3.3. *Divide et impera*

Dividir para conquistar é uma estratégia amplamente utilizada em computação quando temos um problema grande e complexo para resolver. As raízes da expressão, originalmente adaptada do latim Divide et impera, constituía uma forma de estratégia

militar utilizada pelos romanos, onde pequenas e independentes unidades militares eram dividas estrategicamente para conquistar o inimigo (CARROL, 2001). Desta forma, a partir desta abordagem, institui-se que um problema complexo para ser resolvido deve ser partido em problemas menores, de maior facilidade de resolução, onde cada solução encontrada deverá ser combinada, formando a solução para o problema original.

Esta estratégia foi utilizada na concepção da abordagem proposta. Assim, o complexo problema de rastrear a evolução estrutural de um determinado projeto de software foi partido nos seguintes problemas menores e independentes:

P1. Como comunicar com as ***Fonte de Dados***?

P2. Como representar as informações de projeto?

P3. Como transformar as informações de projeto da representação utilizada pela ***Fonte de Dados*** para a representação interna escolhida?

P4. Como comunicar com os ***Visualizadores***?

P5. Como transformar as informações de projeto da representação interna para a representação utilizada pelo ***Visualizador***?

P6. Como manter a representação no ***Visualizador*** constantemente atualizada?

Desta forma, a solução de cada problema apresentado foi combinada de forma a compor a funcionalidade proposta. A seguir, a solução para cada problema exposto anteriormente será apresentada. Note que todas as soluções foram pensadas levando em consideração os requisitos apresentados no inicio do capítulo.

3.3.1. Conectores (Problemas P1 e P4)

Em função da similaridade destes problemas, isto é, em ambos os casos é necessário algum tipo de comunicação com sistemas externos, uma única solução foi

adotada. Componentes conectores independentes realizarão as atividades de comunicação com sistemas externos. Desta forma, o elemento Núcleo apresentado anteriormente como o único componente da abordagem, será decomposto em três componentes distintos, conforme ilustrado na

Figura **11**.

Portanto, haverá um componente Conector de Fonte de Dados que será responsável pela comunicação com as Fontes de Dados externas. Suas responsabilidades incluem resgatar as informações de projeto, converter estas informações da representação externa (i.e. representação da Fonte de Dados) para a representação interna (vide solução para o problema P2) e, por fim, informar ao componente Núcleo da existência destas informações. Note que, este componente terá as mesmas responsabilidades descritas anteriormente, menos as responsabilidades de comunicação e transformações de formato.

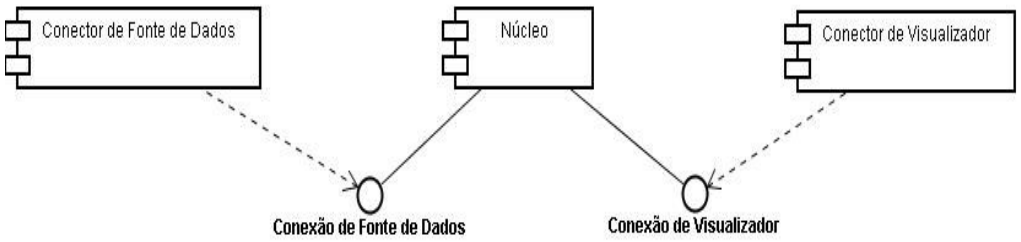

Figura 11. Esquema de utilização de conectores.

Adicionalmente, pode ser observada a existência de um componente Conector de Visualizador. Assim como o Conector de Fonte de Dados, sua funcionalidade principal pode ser vista como a de comunicação com um elemento externo ao sistema, neste caso, um programa ou interface visualizadora (i.e. Visualizador). Especificamente falando, este conector será responsável por realizar uma transformação entre o formato padrão interno para o formato específico de saída.

Desta forma, a partir da mesma fonte de dados poderíamos ter diversos tipos de visualização das informações de projeto. Isto é, para cada tipo de visualização teríamos um conector diferente, podendo ser conectado ou desconectado ao Núcleo sem nenhum impacto neste elemento ou em qualquer outro. Por exemplo, poderíamos implementar dois tipos de conectores de Visualizador, um capaz de representar as informações de projeto de forma gráfica e diagramática, e outro que fosse capaz de, a partir das mesmas informações, realizar uma representação textual estruturada.

Assim, pode ser observada que a abordagem proposta, a priori, não impõe qualquer tipo de representação externa das informações de projeto coletadas das fontes de dados, proporcionando uma maior liberdade e flexibilidade em sua utilização.

O Capítulo 4 trará maiores detalhes de como esta infra-estrutura foi implementada e como um desenvolvedor qualquer poderia contribuir com esta abordagem, criando novos conectores para qualquer tipo de Fonte de Dados e tipo Visualizador.

3.3.2. UML (Problema P2)

O problema P2 diz respeito a como representar as informações de projeto coletadas a partir das fontes de dados. Para tratar este problema, algumas abordagens poderiam ser utilizadas. Por exemplo, poderíamos criar um modelo capaz de representar todos os possíveis artefatos de um projeto de software, como classes, pacotes, atributos, associações e muitos outros.

Apesar de parecer uma boa idéia, poderíamos ter, futuramente, dificuldade em estender tal modelo para, por exemplo, rastrear a evolução de casos de uso de um determinado projeto. Além disto, outro ponto negativo desta abordagem seria obrigar que qualquer desenvolvedor interessado em desenvolver um novo Conector de Fonte de Dados conheça este não popular e desconhecido modelo.

Então, em função disto, uma solução mais generalista e popular, capaz de representar informações da estrutura de um determinado projeto, seria preferível. Esta solução foi encontrada na representação UML. Desta forma, será adotado como modelo de dados, ou representação interna, o metamodelo da UML.

Esta abordagem vem sendo utilizada em alguns trabalhos (WADA, SUZUKI, 2005) (WENZEL, 2005) cujo objetivo é o armazenamento de elementos de projeto capazes de serem modelados via UML. Desta forma, pode ser dito que, com a adoção desta abordagem, é possível identificar benefícios como potencial compatibilidade com diversos sistemas externos de visualização, flexibilidade de customização para um domínio específico, visibilidade e redução da complexidade arquitetural da abordagem proposta, uma vez que não será preciso especificar um modelo próprio de representação.

Adicionalmente a esta utilização do metamodelo da UML, a representação interna também será composta por um modelo de controle. Este será responsável por representar informações não estruturais do projeto, como autoria de uma dada evolução e data da mesma.

3.3.3. Formato Externo => Formato Interno (Problema P3)

A transformação das informações de projeto do formato externo, específico da fonte de dados, para o formato interno (UML) será realizada pelo Conector de Fonte de Dados. Esta transformação poderá ser efetuada a partir de duas maneiras. A primeira é utilizando uma API de criação de elementos UML fornecida pelo componente Núcleo. A segunda opção corresponde a criação dos elementos UML pelo próprio conector. Isto é, a partir de um modelo criado pelo Núcleo, o desenvolvedor do conector poderá criar por conta própria os elementos UML pertinentes e, posteriormente, anexá-los ao modelo criado.

Desta forma, para que um desenvolvedor possa criar uma conexão com um novo tipo de fonte de dados, este deverá conhecer apenas a interface de comunicação com o Núcleo e a forma de criação de elementos UML para realizar as devidas transformações. A Figura 12 ilustra o processo de transformação.

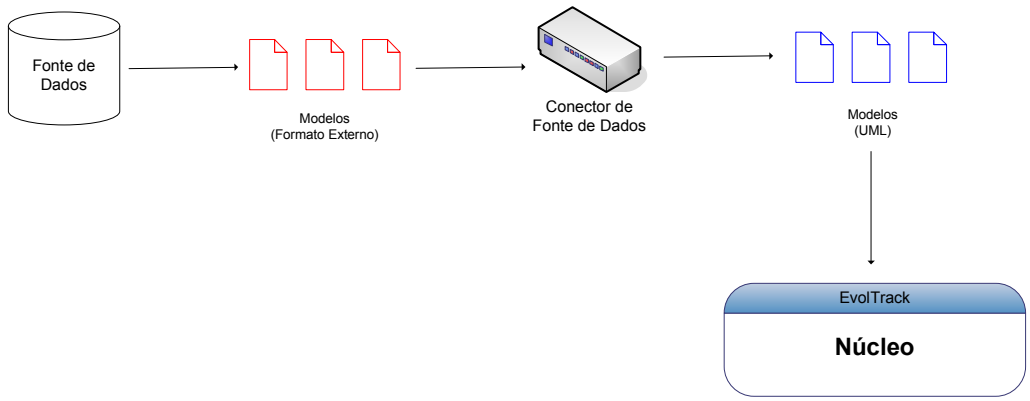

Figura 12. Processo de transformação do formato externo para o formato interno.

Porém, é importante notar que diversas informações coletadas a partir de uma fonte de dados podem resultar em apenas um único modelo na outra ponta da transformação. Isto porque dependerá da política de criação de modelos imposta pelo Conector de Fonte de Dados. Por exemplo, se o conector julgar que qualquer modificação no projeto, por menor que seja, representa uma nova configuração do projeto, então, cada nova informação coletada provavelmente resultará em um novo modelo de projeto. Entretanto, se for política do conector consolidar todas as modificações diárias em apenas um único modelo, então, um grande conjunto de informações coletadas deverá ser utilizado para gerar um único modelo diário.

3.3.4. Padrão de Projeto: Observer (Problema P6)

O problema P6 faz referência a que estratégia deverá ser adotada de forma que o componente Visualizador mantenha-se constantemente atualizado com as informações coletadas da fonte de dados. É importante notar que esta coleta ocorre, em geral, de forma

ativa pelo Conector de Fonte de Dados. Isto é, periodicamente o Conector de Fonte de Dados realiza um processo de comunicação com a fonte de dados à procura de novas informações pertinentes. Quando estas informações são encontradas, realiza-se o processo de transformação descrito anteriormente. Entretanto, nada impede que, caso a Fonte de Dados disponibilize um recurso de registro de interessados em informações, o conector, após ter feito este registro, receba de forma passiva tais informações. Então, automaticamente durante este processo de criação de novos modelos, acontece por parte do Núcleo um processo de aviso de todos os componentes Visualizador conectados.

Assim, pode ser observado que neste caso utilizou-se o padrão de projeto Observer (GAMMA et al., 1994) para que qualquer Visualizador conectado ao Núcleo fosse capaz de receber as atualizações de modelos no instante em que estes fossem criados.

A

Figura **13** apresenta de forma diagramática como este padrão foi utilizado na abordagem proposta. Note que qualquer Conector de Visualizador deverá, em tempo de inicialização, cadastrar-se como um ouvinte do Núcleo, caso necessite manter-se constantemente atualizado.

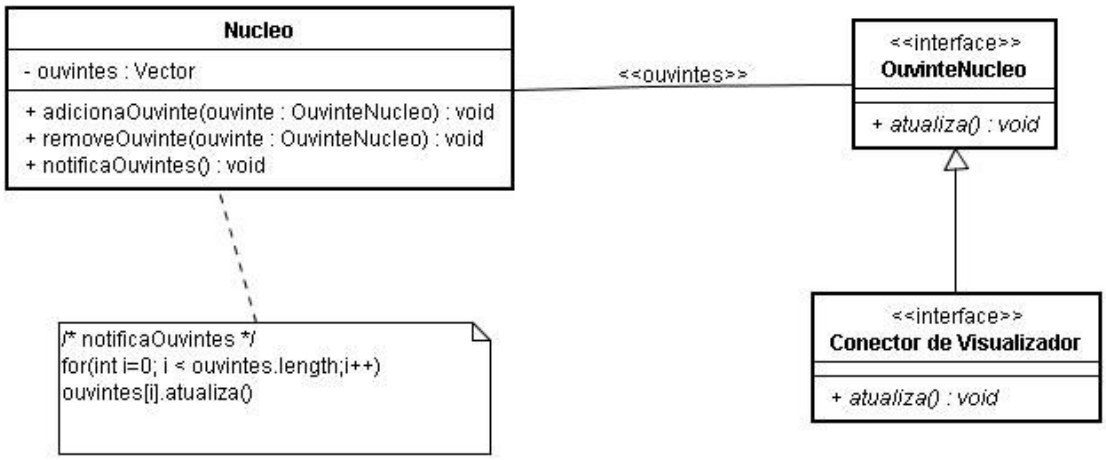

Figura 13. Implementação do padrão de projeto *Observer* na abordagem.

3.3.5. Formato Interno => Formato Externo (Problema P5)

Analogamente ao problema P3, este problema diz respeito a mais um tipo de conversão de formato ou representação. Entretanto, neste caso, refere-se a transformação da representação interna das informações do projeto, armazenadas na forma de modelos UML, para uma representação externa dependente do tipo de visualização que deverá ser utilizado.

A Figura 14 apresenta o processo de transformação de um modelo representado internamente com UML para uma representação específica das informações do projeto em questão.

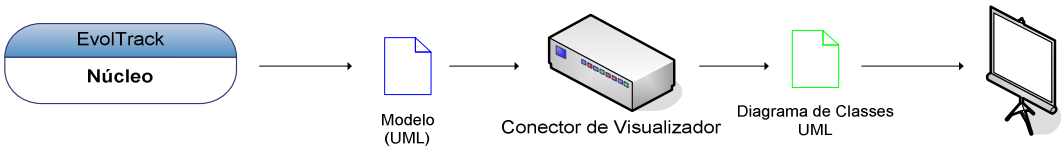

Figura 14. Ilustração do processo de conversão do formato interno para um diagrama.

Note que, neste caso, utiliza-se um diagrama de classes UML como forma de representação externa das informações. Portanto, teremos uma representação diagramática, através da linguagem UML, das informações do projeto. Entretanto, conforme foi apresentado anteriormente, para que outro tipo de representação externa possa ser utilizada, basta que um novo Conector de Visualizador apresente as informações de uma outra forma.

A idéia é poder proporcionar ao usuário, através desta abordagem, a possibilidade de escolha entre diferentes estratégias de apresentação e até mesmo entre diferentes meios físicos de visualização. Isto é, com a construção dos conectores apropriados, esta

abordagem poderia ser utilizada para, ao mesmo tempo, apresentar as informações de projeto com o passar do tempo no monitor conectado ao computador do usuário e apresentar as mesmas informações, possivelmente em uma outra representação, por exemplo, no celular do mesmo usuário ou de outro interessado em obter informações atualizadas sobre o projeto.

Assim, além de proporcionar uma forma pela qual desenvolvedores em um projeto de software possam entender, sob uma determinada perspectiva, como o projeto evoluiu ao longo do tempo, também pode proporcionar a gerentes ou líderes de projeto informações on-the-fly do andamento do projeto. Por exemplo, pode ser facilmente observado quais desenvolvedores estão contribuindo ativamente para o projeto, em qual o período do tempo ocorre a maior parte das evoluções no software, se o software que está sendo desenvolvido está aderente a um modelo de projeto previamente construído, dentre outros.

3.4. Modelo Conceitual

A

Figura **15** exibe o modelo de classes conceitual da abordagem, utilizando a notação UML. O modelo apresenta somente as classes (desconsiderando seus atributos e métodos). Um modelo mais detalhado será apresentado no próximo capítulo.

A abordagem EvolTrack foi modelada tendo em vista os requisitos apresentados e a flexibilidade de utilização. As classes conceituais envolvidas na construção deste trabalho são:

- **Controlador**: Classe responsável por orquestrar o fluxo de mensagens entre os conectores de ***Fonte de Dados*** e os conectores de ***Visualizador***.

Adicionalmente, tem a responsabilidade de manter o estado atual do sistema, persistindo e armazenando informações pertinentes a execução;

- **Projeto**: Define a estrutura de um projeto a ser analisado e provê métodos para a sua manutenção;
- **Ciclo de Vida**: Representa o ciclo de vida de um projeto. Isto é, é o ponto de acesso para todas as informações ao longo do tempo de um determinado projeto criado na abordagem;
- **Gerenciador de Fonte de Dados**: Tem a responsabilidade de descobrir e gerenciar todas as fontes de dados (representadas pelos conectores de fonte de dados) conectadas ao componente ***Núcleo***;

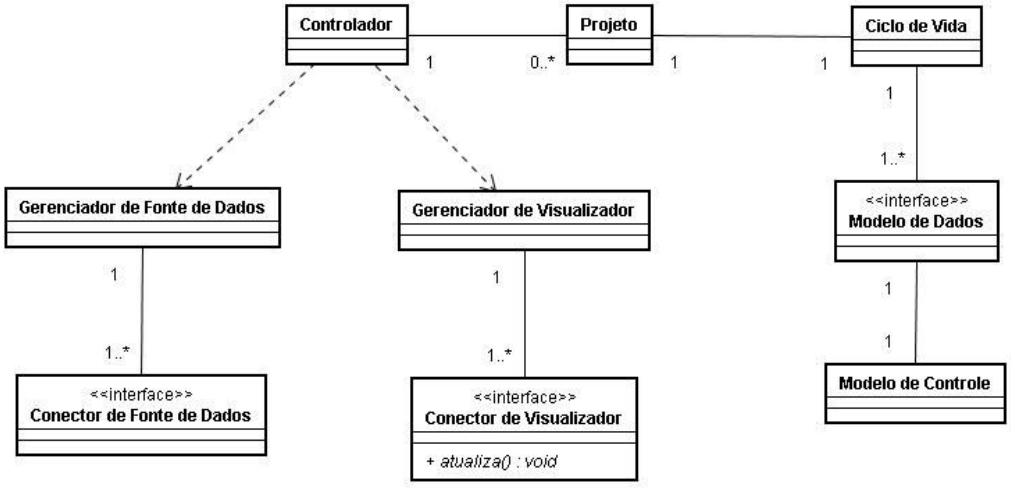

Figura 15. Modelo de Classes Conceitual da abordagem.

- **Gerenciador de Visualizador**: Tem a responsabilidade de descobrir e gerenciar todos os visualizadores (representados pelos conectores de ***Visualizador***) conectados ao componente ***Núcleo***;
- **Conector de Fonte de Dados**: Representa uma fonte de dados específica. Para que uma nova fonte de dados possa ser utilizada, uma nova

implementação desta interface deverá ser criada;

- **Conector de Visualizador**: Analogamente, representa um novo tipo de visualização a ser utilizado. Assim, para que um novo tipo de visualização possa ser utilizado, uma nova implementação desta interface deverá ser criada;
- **Modelo de Dados**: Representa o modelo de dados adotado pela abordagem, neste caso UML. Cada instância de sua implementação atuará como "fotos" do projeto que foram resgatadas ao longo do tempo;
- **Modelo de Controle**: Representa o modelo de controle adotado pela abordagem. Este será responsável por armazenar informações complementares, como a data de uma dada evolução do projeto e seu respectivo responsável.

3.5. Conclusão

O capítulo apresentou uma abordagem de solução que atendesse a todos os requisitos previamente enumerados e apresentados no início do capítulo. Neste contexto, apresentou-se um conjunto de problemas que, ao serem solucionados individualmente, proporcionaram uma solução completa para os requisitos levantados.

Assim, a abordagem proposta apresenta um método sistemático capaz de coletar informações de projetos de software, a princípio através de qualquer fonte de dados; armazenar estas informações em uma notação (UML) amplamente conhecida pela comunidade; e, através desta, disponibilizar ao usuário final uma representação da evolução do software sob análise. Note que, assim como na fonte de dados, a abordagem não se atém a uma forma (por exemplo, diagramática) e nem a um sistema específico de

visualização.

No Capítulo 4 é descrita uma implementação para a abordagem proposta, neste caso, apresentando uma escolha de fonte de dados e sistema de visualização específicos, concretizando os conceitos aqui apresentados.

Capítulo 4. Projeto e Implementação

4.1. Introdução

O capítulo 3 apresentou de forma funcional a abordagem proposta por este trabalho. Este capítulo tem por objetivo apresentar sob a perspectiva técnica o projeto e a implementação dos pontos discutidos anteriormente. Isto é, são abordados os detalhes de como cada conceito discutido e definido pela abordagem proposta pode ser implementado através de um sistema de software.

Portanto, fez-se necessário o projeto e implementação de um conjunto de ferramentas. Estas estão dividas entre os blocos funcionais da abordagem: Fonte de Dados, Núcleo e Visualizador. Desta forma, no que tange ao desenvolvimento deste projeto, as seguintes ferramentas foram construídas: EvolTrack-LH, EvolTrack-Kernel, EvolTrack-Eclipse.

Este capítulo está dividido da seguinte forma: a seção 4.2 apresenta a infra-estrutura Eclipse utilizada como base para a implementação das ferramentas; a seção 4.3 apresenta o projeto e implementação da ferramenta EvolTrack-Kernel, além de evidenciar a forma de comunicação entre esta e as demais ferramentas criadas; a seção 4.4 apresenta o projeto e implementação da ferramenta EvolTrack-LH, componente criado para interagir com a ferramenta externa Lighthouse; a seção 4.5 apresenta o projeto e implementação da ferramenta EvolTrack-Eclipse; na seção 4.6 é apresentado o conjunto de ferramentas em utilização; a seção 4.7 exibe os principais pontos

apresentados e conclui o capítulo.

4.2. Infra-estrutura Eclipse

A ferramenta Eclipse não corresponde a um programa monolítico, mas sim a um *Kernel* reduzido, também conhecido como *Plug-in Loader*, cercado por centenas de *plug-ins* (CLAYBERG e RUBEL, 2006). A Figura 16 apresenta resumidamente esta arquitetura. Note que, a plataforma corresponde a um pequeno conjunto de componentes que cooperam entre si para formar a funcionalidade básica da mesma e que, adicionalmente, montam um arcabouço estrutural para a conexão de novas funcionalidades na forma de *plug-ins*.

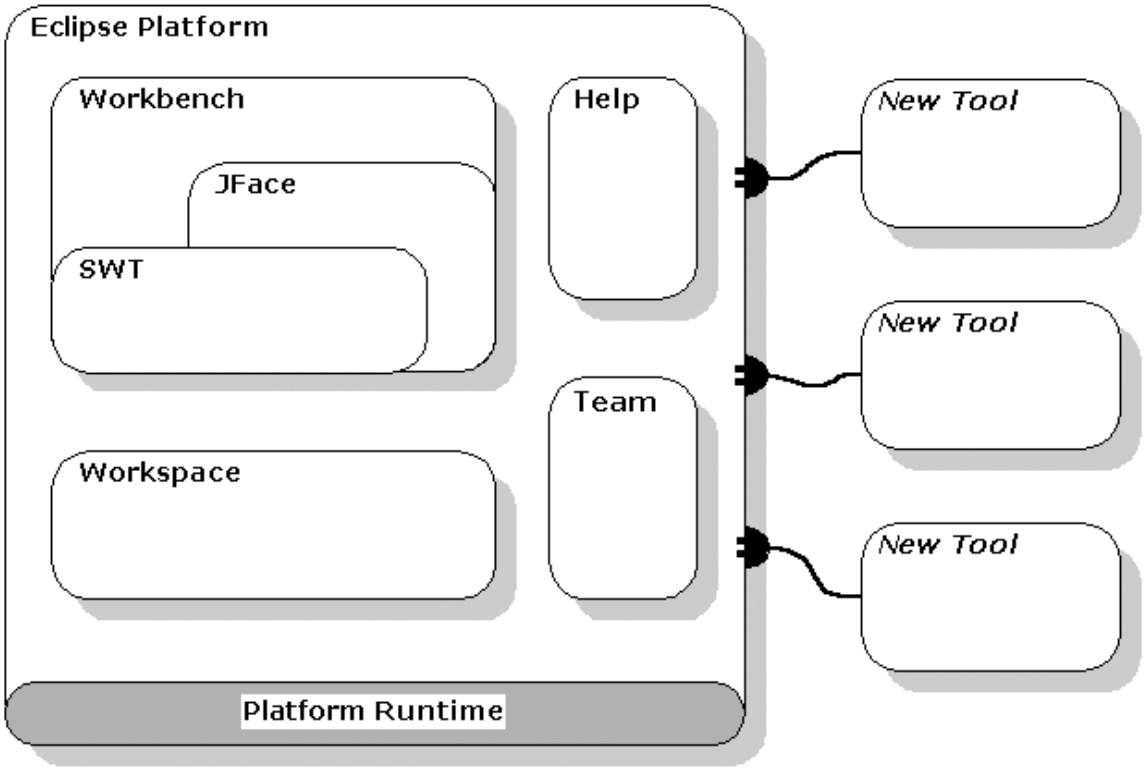

Figura 16. Esquema que representa resumidamente a plataforma Eclipse.

Neste contexto, de forma a viabilizar esta conexão de novas funcionalidades, ou

plug-ins, a seguinte abordagem foi adotada: dois arquivos, MANIFEST.MF (vide Figura 17) e plugin.xml (vide Figura 18), devem ser criados e disponibilizados para a plataforma. O primeiro, serve para que o desenvolvedor do plug-in especifique informações como a versão do mesmo, as relações de dependência com outros plug-ins, informações de visibilidade (i.e. que partes do plug-in poderão ser acessadas por outros plug-ins da plataforma), informações de carga, dentre outras. Por exemplo, na Figura 17 podem ser observados através do atributo "Export-Package" todos os pacotes que este plug-in permitirá acesso externo. Desta forma, pacotes que não estejam listados neste atributo não poderão ter seus recursos acessados por outros plug-ins instalados no Eclipse, facilitando, portanto, a utilização de técnicas de encapsulamento e ocultação da informação pelos desenvolvedores destes sistemas.

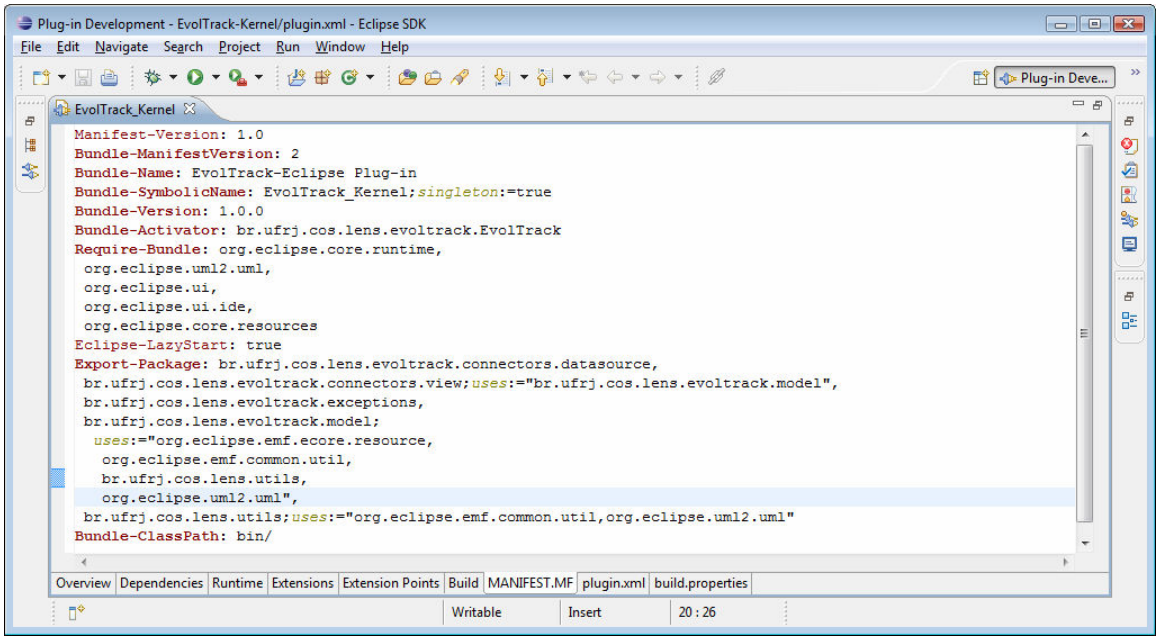

Figura 17. Arquivo MANIFEST.MF utilizado pelo plug-in EvolTrack-Kernel.

Já o segundo arquivo, plugin.xml, contém a descrição propriamente dita do plug-in. Isto é, a partir deste, arquivo a plataforma obtém o ponto de entrada do plug-in e o

conjunto de funcionalidades que este deverá fornecer para o restante da plataforma. No exemplo apresentado pela Figura 18, observa-se que o ponto de entrada do plug-in em questão é representado pela classe “br.ufrj.cos.evoltrack.EvolTrack”. Para tal, esta classe deve obrigatoriamente estender a classe abstrata Plugin ou alguma subclasse da mesma, como por exemplo, a classe abstrata AbstractUIPlugin, utilizada por plug-ins que de alguma forma estenderão as funcionalidade de interface com o usuário da plataforma Eclipse.

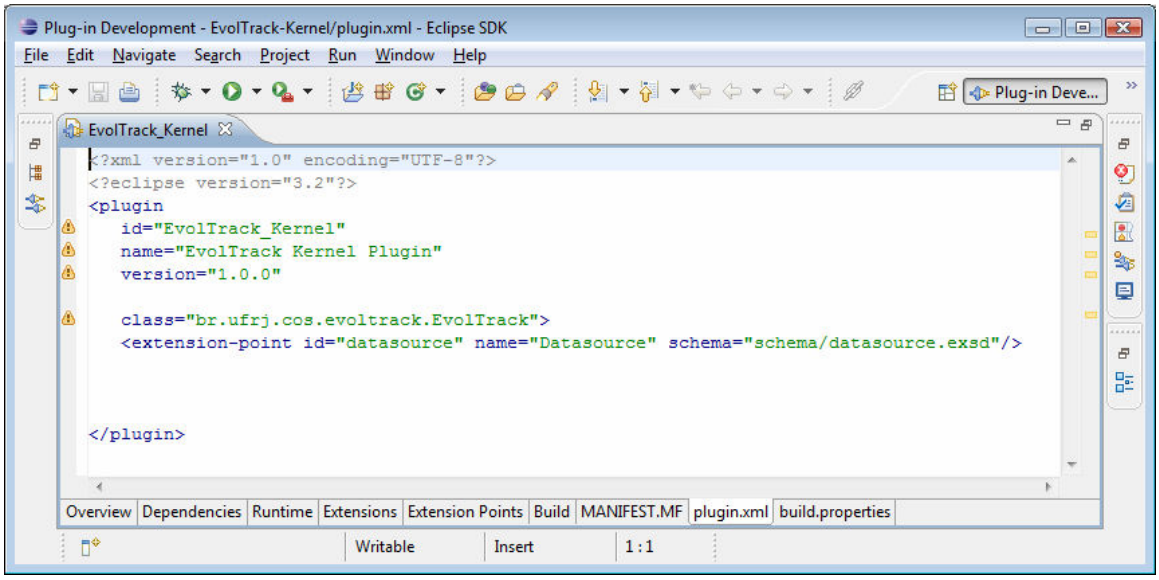

Figura 18. Arquivo PLUGIN.XML utilizado pelo plug-in EvolTrack-Kernel.

Portanto, a partir da infra-estrutura supracitada, o conjunto de funcionalidades propostas por este trabalho foi implementado. Desta forma, três novos plug-ins Eclipse foram desenvolvidos, cada um correspondendo a um módulo funcional descrito no Capítulo 3. Ou seja, um plug-in para representar o módulo Núcleo, outro para o módulo Fonte de Dados e outro para o módulo Visualizador.

4.3. EvolTrack-Kernel

A partir dos principais conceitos desenvolvidos no capítulo 3 que desempenharam um papel central na arquitetura (i.e. Projeto que será evoluído, Ciclo de Vida, Modelo, dentre outros), o seguinte diagrama de classe, apresentado na Figura 19, foi proposto para o módulo Núcleo e implementado na forma de um novo plug-in Eclipse, EvolTrack-Kernel. Note que, este diagrama representa uma instanciação para o modelo conceitual descrito no Capítulo 3.

Note que, a classe Kernel representa um papel central neste projeto de implementação. Esta é responsável por gerenciar os projetos, ainda em forma de modelo, que serão analisados posteriormente pelo usuário do módulo Visualizador. Como pode ser observada, a classe Kernel pode estar associada a diversos projetos. Cada projeto está associado a um ciclo de vida, o que no projeto corresponde a classe Lifecycle. Esta, por sua vez, estará associada a diversos modelos UML, onde cada modelo representa uma versão, ou "fotografia", do projeto em questão. Neste contexto, é importante reparar que cada modelo UML citado será materializado por uma classe que implementa a interface Model do projeto UML2 (UML2, 2008) da fundação Eclipse.

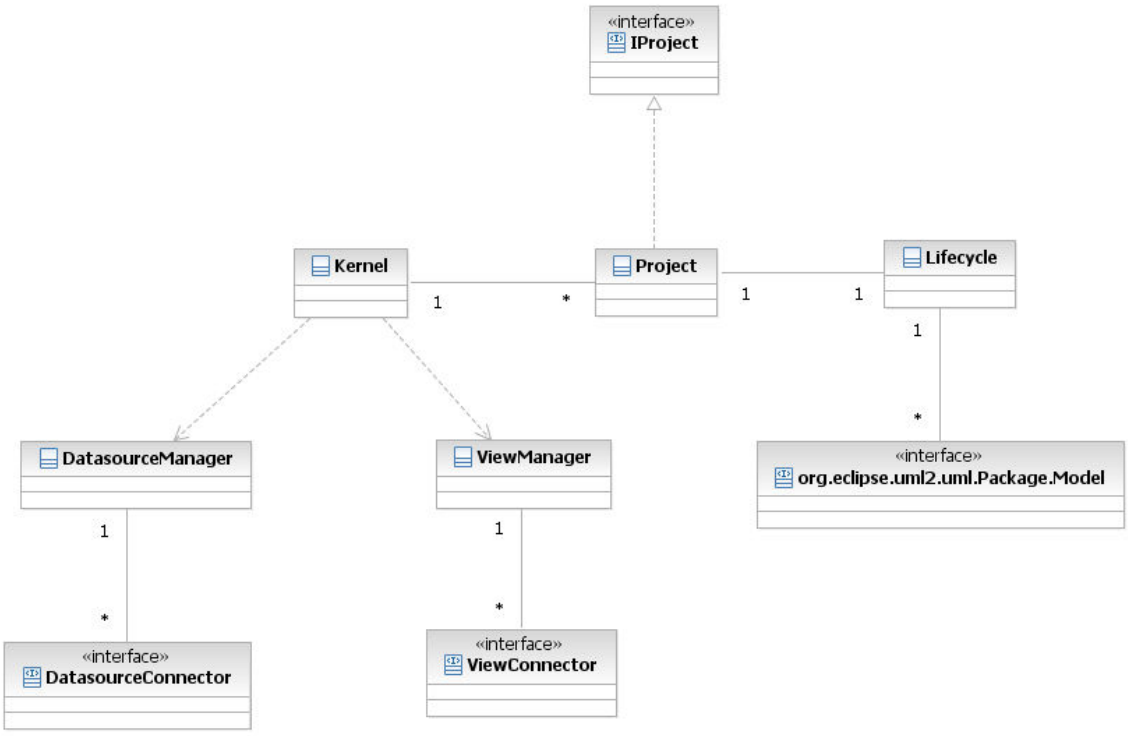

Figura 19. Diagrama de Classe UML com as principais classes do EvolTrack-Kernel.

Além disto, vale ressaltar a importância das classes DatasourceManager e ViewManager. Estas representam o ponto de entrada para os demais módulos da arquitetura, Fonte de Dados e Visualizador. Isto é, para que o plug-in EvolTrack-Kernel possa tomar conhecimento dos demais plug-ins presentes em uma determinada configuração, estes devem, por sua vez, cadastrar-se como representantes de Fonte de Dados, através da classe DatasourceManager, ou como Visualizadores, através da classe ViewManager.

Adicionalmente, para viabilizar a comunicação do Núcleo com as Fontes de Dados e os Visualizadores, duas interfaces foram definidas: DatasourceConnector e ViewConnector. Desta forma, uma implementação do módulo Fonte de Dados deverá, além de se cadastrar no Núcleo, conforme descrito anteriormente, implementar a interface DatasourceConnector. Analogamente, uma implementação do módulo

Visualizador deverá implementar a interface ViewConnector.

Portanto, esta abordagem possibilita a criação de diferentes implementações para Visualizadores e Fontes de Dados. Desta forma, no contexto deste trabalho, desenvolveu-se uma implementação para cada módulo deste. Para o módulo de Fonte de Dados, implementou-se o plug-in EvolTrack-LH, responsável pela interação com o sistema Lighthouse. Já para o módulo Visualizador, implementou-se o plug-in EvolTrack-Eclipse, que é responsável por apresentar as informações, de forma diagramática, na própria plataforma Eclipse.

4.4. EvolTrack-LH

O plug-in EvolTrack-LH (vide Figura 20) foi desenvolvido para viabilizar a conexão entre o projeto EvolTrack como um todo e o sistema Lighthouse. Este, por sua vez, gera para cada evento de um desenvolvedor na plataforma Eclipse, como por exemplo a criação de uma nova classe ou um novo método, um registro na base de dados do próprio sistema.

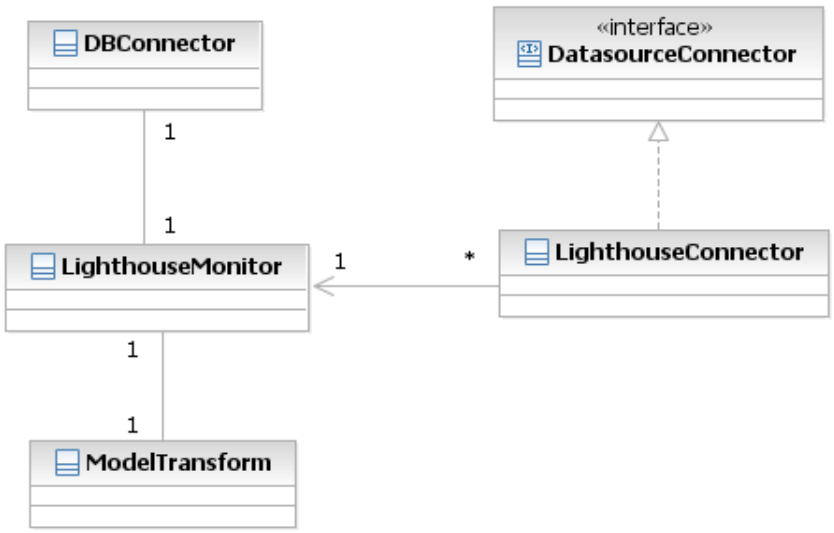

Figura 20. Diagrama de Classe UML com as principais classes do EvolTrack-LH.

Desta forma, o projeto de implementação explicitado na Figura 20 tem por objetivo extrair todos os eventos pertinentes a um dado projeto de desenvolvimento da base de dados do Lighthouse e traduzi-los para a forma de representação interna disponibilizada pelo Núcleo. Isto é, neste caso, os diversos eventos serão traduzidos em novas versões de modelo do projeto, representadas pela interface Model fornecida pela iniciativa UM2, conforme descrito na seção 4.3.

A classe responsável por esta conversão é a ModelTransform. Cada vez que um novo evento é detectado pela classe LighthouseMonitor, uma transformação é realizada e um novo Modelo para este projeto é informado ao Núcleo, através da classe DatasourceManager. Este será responsável por incrementar de uma versão o ciclo de vida do projeto em análise e informar todos à todos os Visualizadores pertinentes de que uma nova atualização no projeto foi realizada.

4.5. EvolTrack-Eclipse

Como infra-estrutura para a atividade de apresentação da evolução modelo a modelo construída pelo restante do sistema, optou-se pela utilização da plataforma Eclipse. A proposta deste trabalho visa o apoio ao engenheiro de software e outros interessados no entendimento do processo de evolução de um determinado projeto. Como a plataforma Eclipse atualmente é, amplamente, utilizada por este conjunto de usuários, concluiu-se que a utilização da mesma para apoiar o processo de desenvolvimento poderia ser proveitosamente estendida para apoiar a compreensão do ciclo de vida do projeto em desenvolvimento.

Desta forma, a partir de uma única ferramenta, obtém-se um ferramental integrado para os processos de desenvolvimento do software e entendimento estrutural de sua evolução ao longo do tempo. Com isso, seguindo a mesma linha de raciocínio anterior, foi especificado e implementado um plugin (vide Figura 21), EvolTrack-Eclipse, capaz de interagir com o módulo Núcleo e, a partir de suas informações providas, apresentar de forma diagramática cada modelo gerado pelo projeto (i.e. cada "foto" do projeto) ao longo do seu ciclo de vida.

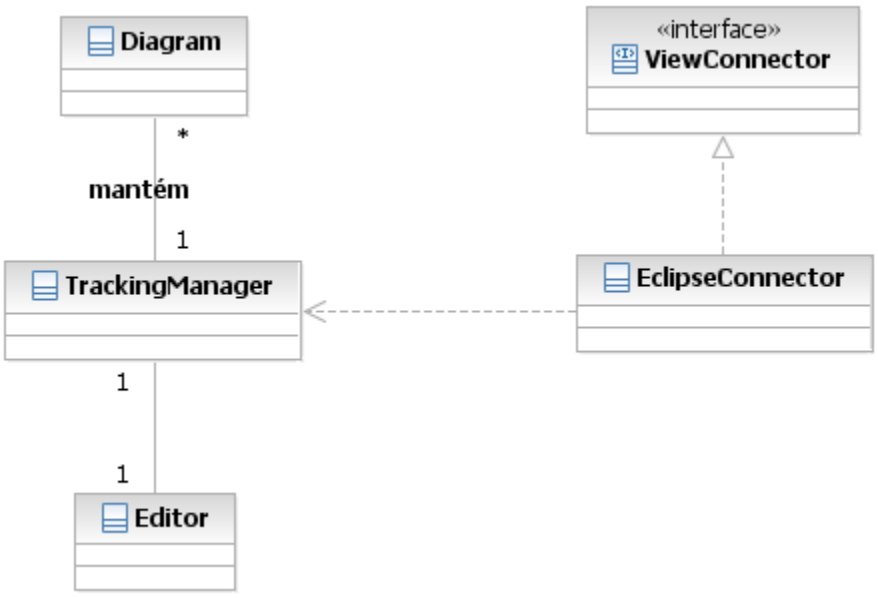

Figura 21. Diagrama de classe UML com as principais classes do EvolTrack-Eclipse.

Como pode ser observada, a classe EclipseConnector implementa a interface ViewConnector definida pelo módulo Núcleo, que caracteriza este plugin como sendo do tipo Visualizador, assim como implementações da interface DatasourceConnector definem módulos Fonte de Dados. Desta forma, é através desta classe (EclipseConnector) que todo fluxo de informação trafegado a partir do módulo Núcleo é tratado. Note que, conforme descrito no capítulo 3, tais classes conectoras implementam o padrão de projeto

Observer. Neste caso, são ouvintes do módulo Núcleo. Portanto, a cada novo evento criado no mesmo (i.e. novo modelo de projeto criado), mensagens de notificação são enviadas para todos os módulos Visualizadores ouvintes do mesmo. Através deste mecanismo assíncrono, o plugin EvolTrack-Eclipse recebe novas informações do projeto, a medida que estas acontecem no tempo. Note que, pelo fato de ser ouvinte do Núcleo e não de projetos específicos, as informações de todos os projetos disponíveis serão recebidas pelo plug-in.

Neste contexto, para cada novo modelo de projeto criado e detectado pela classe EclipseConnector, uma ordem de criação de diagrama é disparada para a classe TrackingManager. Esta é responsável por gerenciar toda e qualquer criação de diagrama, representado pela entidade Diagram no Eclipse, e adicionalmente controlar sua exibição para o usuário, cuja interface é representada e concretizada pela classe Editor.

4.6. Rastreando a Evolução

Como forma de apresentar a utilização da ferramenta, o sistema EvolTrack foi aplicado para acompanhar a evolução de um pequeno projeto, chamado de “Exemplo”, que será construído. Um ponto a ser ressaltado é que a ferramenta foi desenvolvida com o intuito de não ser intrusiva no trabalho dos desenvolvedores que a utilizam. Desta forma, é importante notar que a melhor forma de usufruir dos recursos oferecidos por esta implementação é através da utilização de dois monitores ou através de um grande monitor onde se possa observar simultaneamente mais de uma janela. A Figura 22 ilustra esta recomendação com um exemplo de utilização do sistema Lighthouse apresentado anteriormente.

Figura 22. Exemplo de utilização de dois monitores com o sistema Lighthouse.

Repare que neste caso, enquanto o desenvolvedor concentra-se em suas atividades de desenvolvimento, através do monitor esquerdo, seu sistema perceptivo consegue facilmente notar determinados padrões de variação em um monitor secundário, situado à direita. Entretanto, vale ressaltar que a percepção é uma característica individual de cada desenvolvedor, de modo que, o que pode ser facilmente identificado por um, pode não ser identificado por outro.

Diante disto, suponha que o segundo monitor será utilizado pela ferramenta EvolTrack. Neste caso, a janela Eclipse hospedada no mesmo deverá ser associada à perspectiva criada pelo EvolTrack. Esta mudança de perspectiva é apresentada na Figura 23.

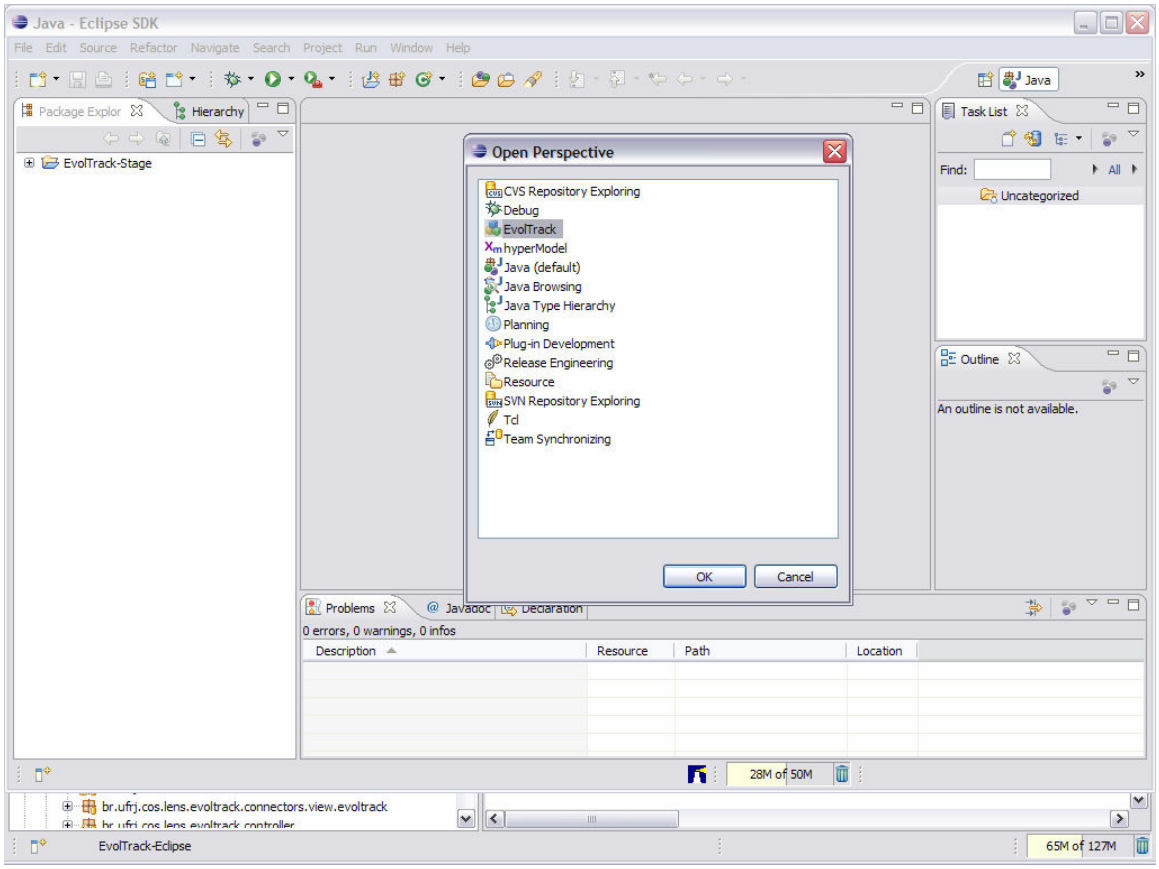

Figura 23. Abertura da perspectiva EvolTrack no Eclipse.

Uma vez efetuada a mudança, o sistema EvolTrack estará apto a ser utilizado. Então, voltando ao exemplo discutido inicialmente, os desenvolvedores do projeto "Exemplo" já efetuaram a criação de no ambiente de programação Java do Eclipse (monitor esquerdo) um pacote Java, chamado de "utilidades", e uma classe inicial chamada de "Arquivo". A Figura 24 apresenta o estado atual do projeto após tais evoluções.

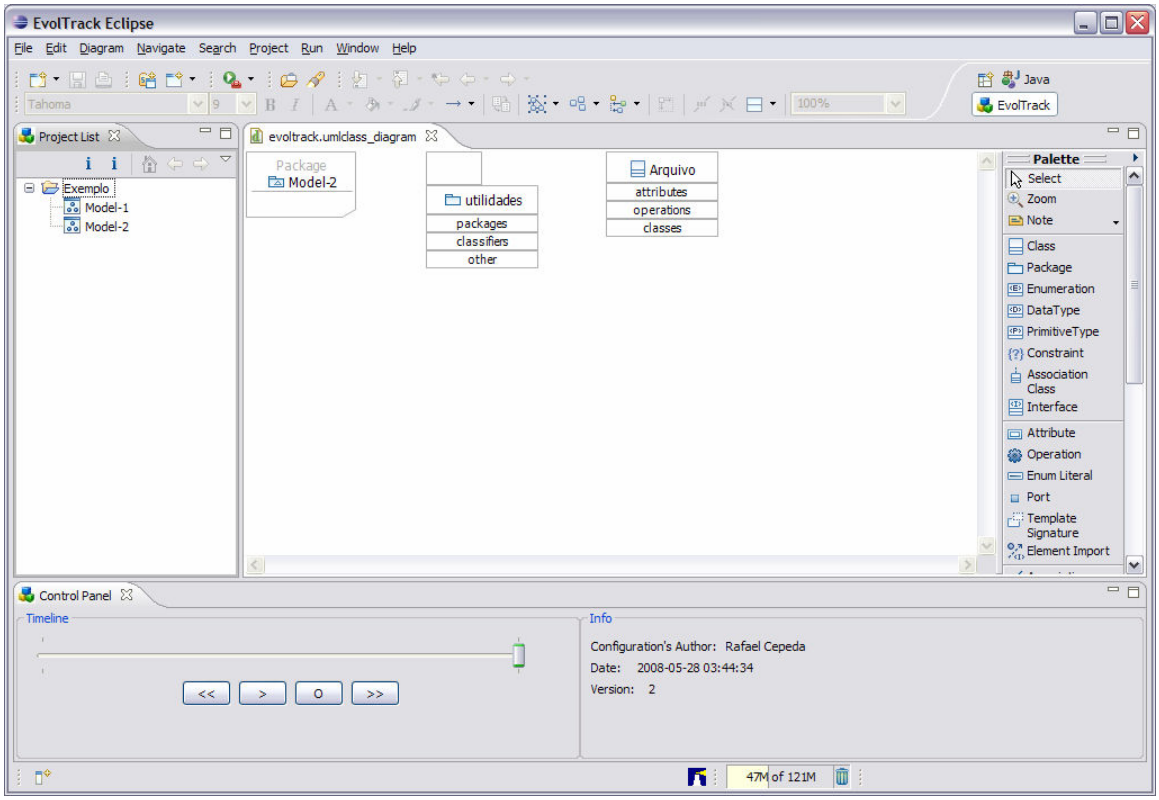

Figura 24. Estado do projeto após a crição de um pacote “utilidades” e uma classe “Arquivo”.

Note que, pelo fato da fonte de dados utilizada, Lighthouse, entender como uma nova versão ou configuração do projeto a simples ocorrência de um único evento que altere sua estrutura, duas versões iniciais foram geradas no exemplo acima. A primeira versão do modelo apresenta exclusivamente o pacote “utilidades” criado. Então, com o advento da criação da classe “Arquivo”, uma nova versão para o modelo do projeto é gerada, resultando na estrutura apresentada na Figura 24.

Antes de dar seqüência no exemplo, vale notar os elementos participantes do cenário sendo executado. Isto é, neste caso, para o papel de fonte de dados está sendo utilizado o sistema Lighthouse, conforme dito anteriormente. Adicionalmente, para o papel de visualizador, o próprio sistema Eclipse está sendo utilizado. Além disto, existem desenvolvedores responsáveis por evoluir o projeto em questão. A Figura 25 sintetiza a

abordagem utilizada.

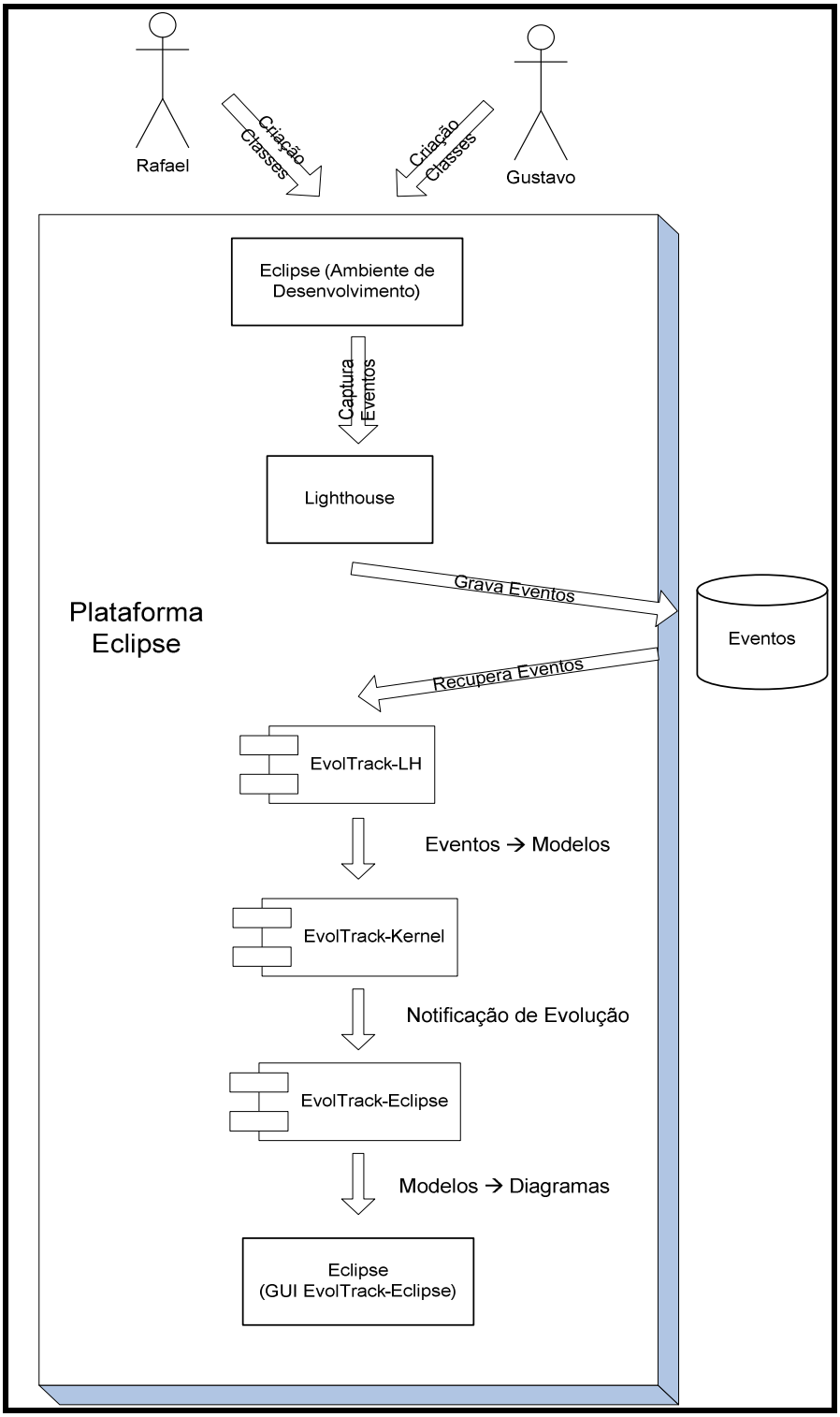

Figura 25. Exemplo de utilização do sistema EvolTrack.

Retornando ao exemplo, desta vez um desenvolvedor do projeto criou uma nova classe chamada de “FTP”, que no caso será responsável por fornecer serviços relacionados a este protocolo. A Figura 26 apresenta o código fonte inicial criado para esta classe. Note que foram adicionados dois atributos na mesma. O primeiro, do tipo “Arquivo”, faz referência à classe previamente criada. O segundo representa apenas uma variável numérica que será utilizada posteriormente no código.

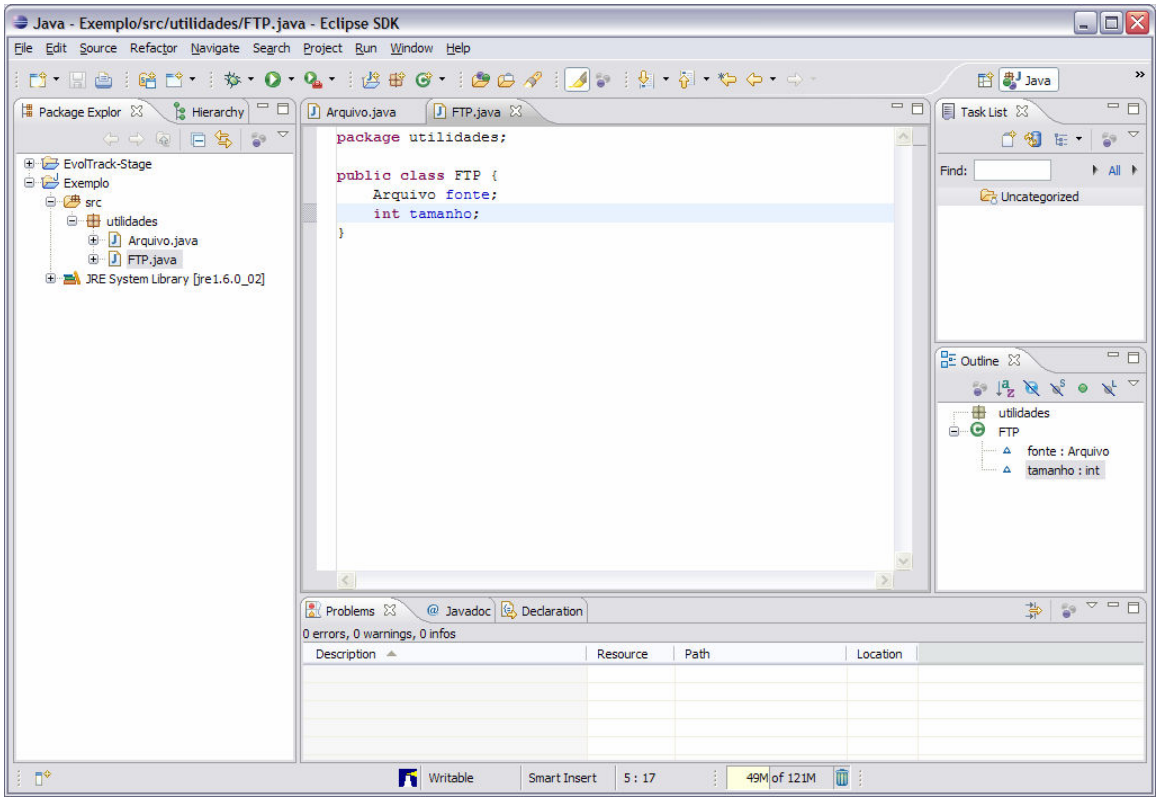

Figura 26. Código fonte da classe “FTP”.

O resultado desta nova implementação de projeto pode ser observado na Figura 27. Repare que não apenas os atributos para classe “FTP” foram criados, mas adicionalmente, um relacionamento de associação também foi criado entre as classes “FTP” e “Arquivo”. A ferremanta EvolTrack suporta todos os tipos de relacionamentos encontrados na UML. Entretanto, vale frisar que é de responsabilidade do desenvolvedor

do programa conector de fonte de dados estabelecer critérios e utilizar adequadamente as funcionalidades oferecidas pelo componente ***Núcleo*** para a criação dos elementos da UML pertinentes.

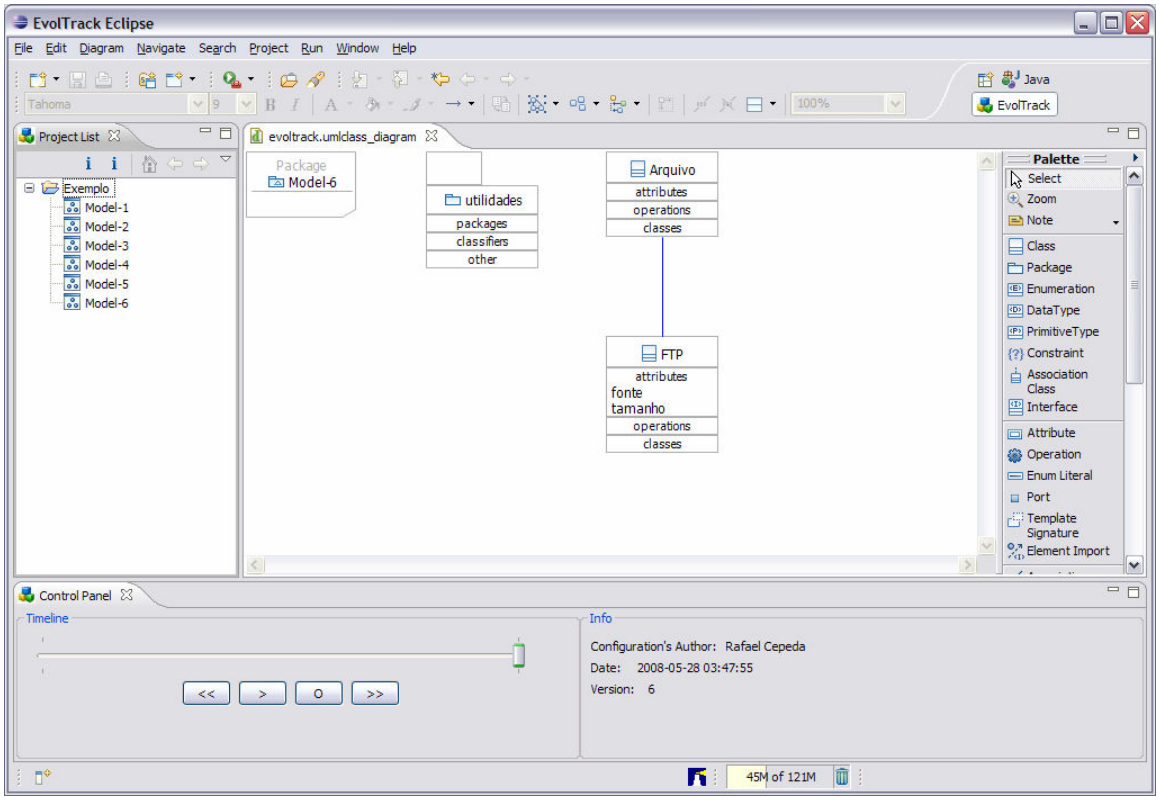

Figura 27. Estado do projeto após a criação da classe “FTP” e de sua codificação inicial.

Finalizando o exemplo apresentado, os desenvolvedores do projeto realizaram uma série de evoluções, como a criação de uma nova classe chamada “Disco”, a criação de um novo atributo desta classe, chamado “espaco”, e a criação de um método de acesso a este atributo criado, chamado de “retornaEspaco”. Além disto, um novo atributo do tipo “Disco” foi criado na classe “FTP”. As três figuras a seguir apresentam esta evolução.

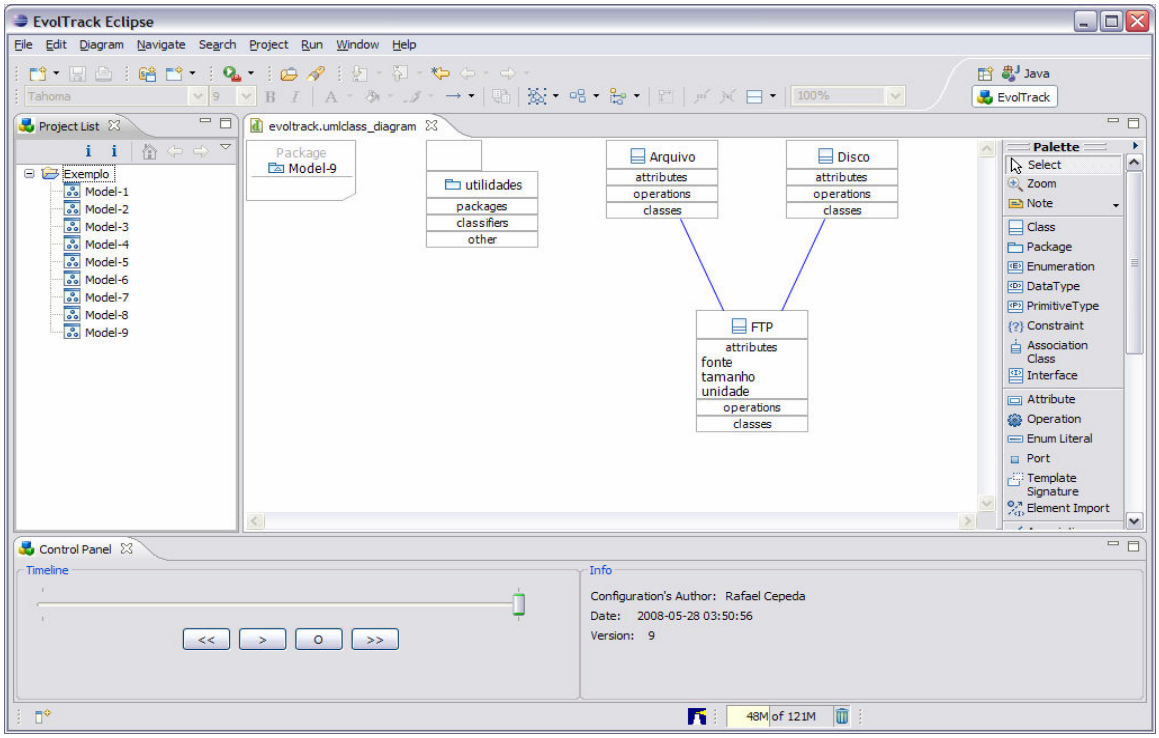

Figura 28. Nova classe incluída no projeto, “Disco”.

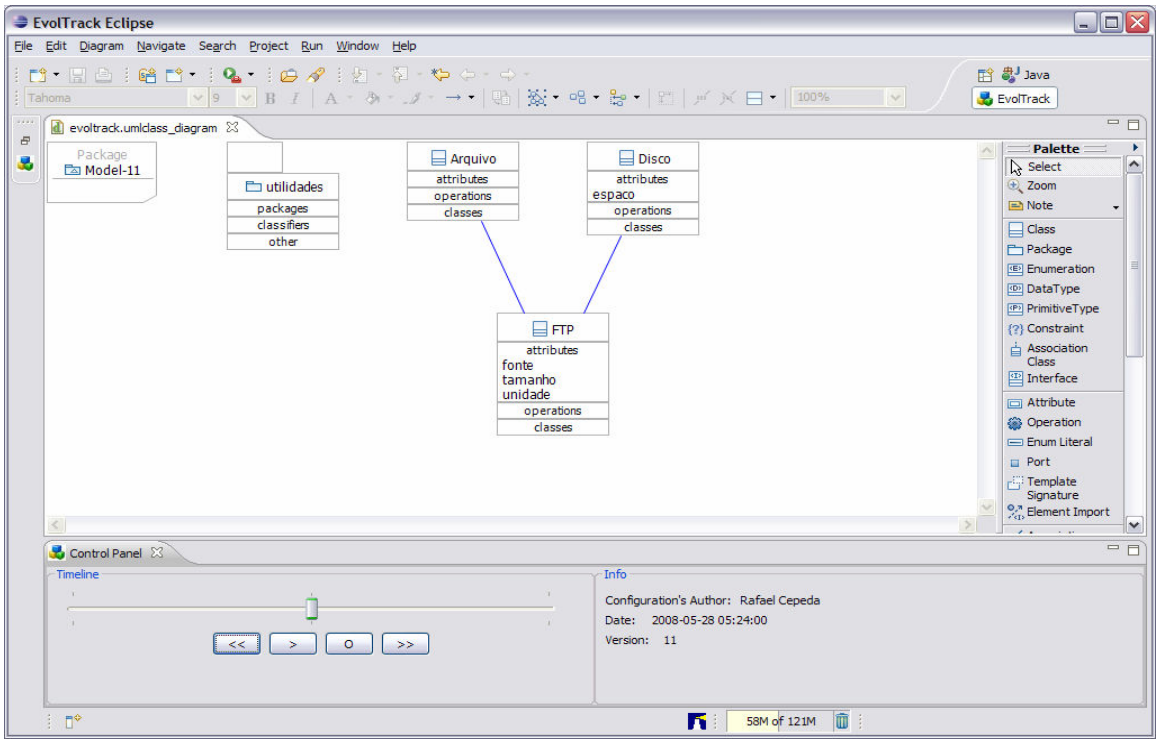

Figura 29. Atributo “espaço” criado na classe “Disco”.

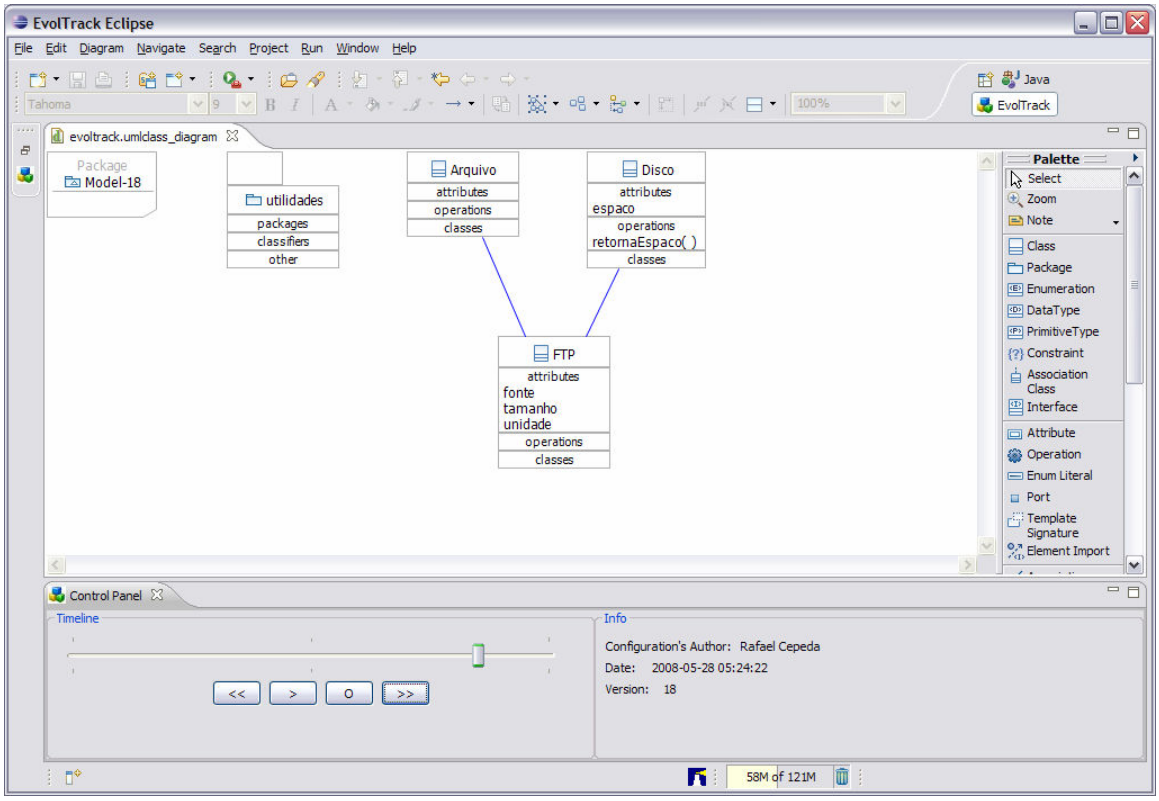

Figura 30. Método "retornaEspaco" adicionada à classe "Disco".

A seguir, são apresentadas as principais funcionalidades e características da ferramenta, disponibilizadas para o usuário. A Figura 31, além de apresentar a última versão do projeto "Exemplo" após diversas evoluções, é divida em 4 áreas funcionais principais: (A), (B), (C), (D).

A área (A) é responsável por apresentar os diagramas gerados pelo EvolTrack. Conforme discutido anteriormente, representa um editor baseado na tecnologia UML2Tools (UML2Tools, 2008). A área (B) apresenta, em forma de lista, todos os modelos gerados até o momento para o projeto em questão. Serve também como meio de acesso rápido para versões específicas do modelo, ou seja, auxilia a navegação entre os modelos gerados.

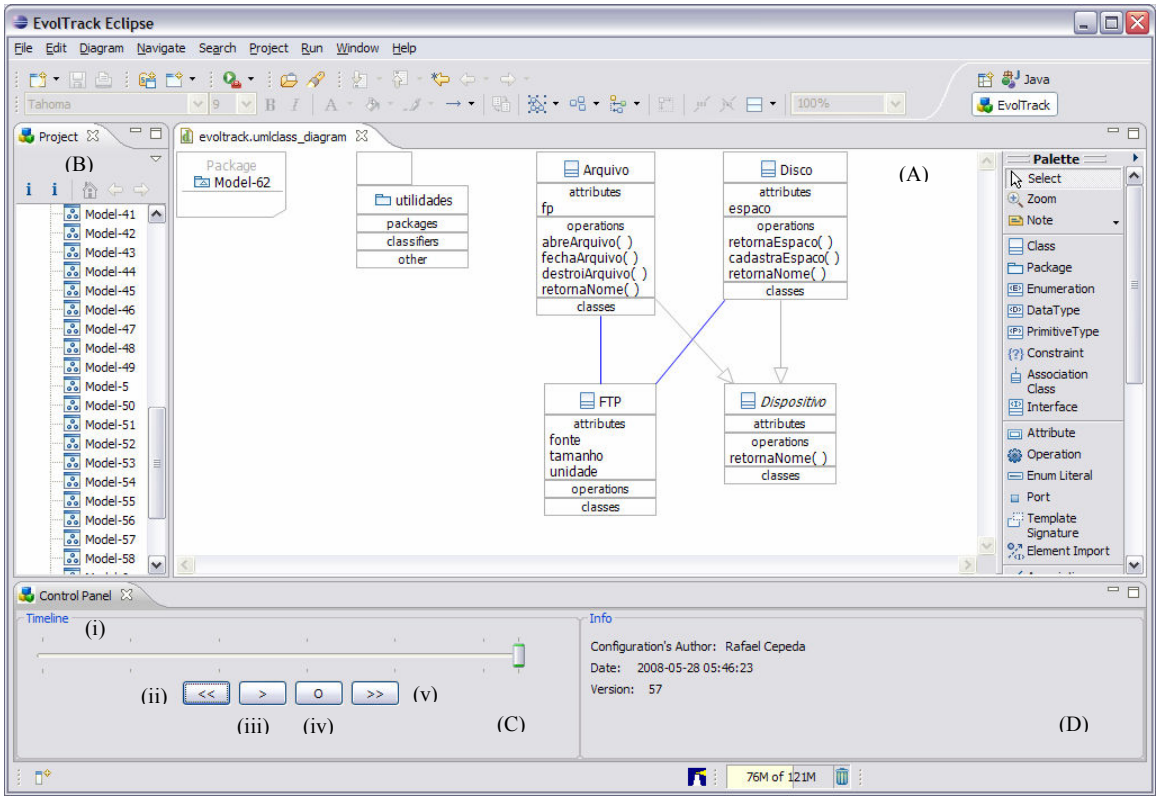

Figura 31. Principais áreas e funcionalidades da interface com usuário do EvolTrack.

A área (C) reúne um conjunto de funcionalidades de navegação extremamente úteis durante o processo de visualização. Neste contexto, a funcionalidade (i) representa a linha do tempo do projeto. O usuário pode livremente rolar a barra para a direita, avançando no tempo, ou para esquerda, retornando no tempo, visualizando, desta forma, todo o ciclo de evolução do projeto. O conjunto de funcionalidades (ii), (iii), (iv) e (v) atuam como em um player musical, porém, neste caso, atuam como um player do projeto. A funcionalidade (ii) permite que o usuário volte um modelo para trás na história, enquanto que a funcionalidade (v) permite que este avance um modelo para frente. A funcionalidade (iii) atua como um play do projeto, ou seja, apresenta automaticamente toda a história da evolução do projeto, modelo a modelo. A funcionalidade (iv) atua conseqüentemente como um stop, isto é, tem a função de parar a execução automática.

A última área, (D), possui um caráter apenas informativo. Através desta, é possível verificar qual é a versão do modelo que está sendo exibida, em que data esta versão do modelo foi criada, e por quem esta versão do modelo foi criada. Note que esta versão do modelo não deve ser confundida com a versão de um sistema de versionamento. Esta versão será um reflexo da política de criação de modelos adotada pela fonte de dados em utilização, ou seja, se um novo modelo é gerado para nova a versão de um sistema de versionamente, então, de fato, este identificador representará a versão do repositório. Porém, caso um modelo seja gerado para cada ação do usuário em seu ambiente de desenvolvimento, conforme explicitado no exemplo acima, então este identificador não terá relação com a versão do projeto em um sistema de versionamento.

4.7. Conclusão

Este capítulo apresentou detalhes do projeto e implementação de todo o sistema EvolTrack para suportar o rastreamento da evolução estrutural de um determinado projeto. Desta forma, foram apresentados as instâncias criadas para os elementos funcionais descritos no Capítulo 3 (i.e. Fonte de Dados, Núcleo e Visualizador).

Neste contexto, com o intuito de dar forma à abordagem proposta, três plug-ins para plataforma Eclipse foram desenvolvidos. O plug-in EvolTrack-LH foi desenvolvido para suportar a interação entre o sistema Lighthouse, capaz de capturar eventos da plataforma Eclipse gerados pelos desenvolvedores, e a implementação do módulo Núcleo. Esta interação inclui, além de outras atividades, a conversão das informações representadas em um formato externo (ex.: registros em uma base de dados) para o formato interno definido pela abordagem, isto é, representação UML.

Estes modelos são armazenados e gerenciados pela plug-in EvolTrack-kernel. Este, adicionalmente, é responsável por notificar todas as implementações para

Visualizadores possivelmente interessadas em receber atualizações deste projeto. Neste trabalho, uma implementação de Visualizadores para o próprio Eclipse foi criada, a partir do plug-in EvolTrack-Eclipse conforme descrito anteriormente.

Desta forma, este capítulo apresentou uma implementação para todos os elementos e conceitos definidos e discutidos no capitulo 3, e, portanto, apresentou um conjunto de ferramentas que, ao cooperarem, proporcionam ao usuário final um meio para que este visualize e entenda, de forma organizada e sistemática, os principais elementos que compõem a estrutura de um projeto em construção.

Capítulo 5. Conclusões

5.1. Contribuições

Na seção 2.8 foi realizado um comparativo entre as ferramentas pesquisadas, frente a um conjunto de critérios estabelecidos, verificou-se que nenhuma das abordagens atendia por completo aos objetivos apresentados no Capítulo 1. A seguir, os critérios são reapresentados:

C1. Tipo de fonte dados utilizada: Indica que outra ferramenta externa armazena os dados utilizados sobre o sistema em análise.

C2. Tipo de visualização utilizada: Indica que tipo de recurso visual foi adotado para representar as características do sistema em análise.

C3. Granularidade das informações apresentadas: Indica o nível de granularidade das informações extraídas do sistema em análise que são apresentadas.

C4. Integração com algum ambiente de desenvolvimento de software: Indica se a ferramenta possui algum tipo de integração com ambientes de desenvolvimento de software, podendo extrair, alterar ou apresentar informações destes ambientes.

C5. Atualização visual automática: Indica se a representação visual apresentada ao usuário sofre atualização automática, conforme novos dados são gerados e extraídos da fonte dados, ou se é preciso manualmente atualizar a representação.

C6. Análise Temporal: Indica se a abordagem em questão proporciona ao seu usuário informações da evolução do sistema sob análise em espaços discretos do tempo, continuamente no tempo, ou ambas as estratégias.

A Tabela 2 exibe novamente o comparativo, porém, incluindo desta vez a abordagem EvolTrack.

Ferramentas						
Critério	**GASE**	**EvoLens**	**CVSscan**	**softChange**	**Augur**	**EvolTrack**
C1	Usuário	CVS	CVS	CVS	Variável	Variável
C2	Diagrama próprio	Diagrama próprio + Lens-View	Gráfico Baseado em Linhas	Gráfico Baseado em *Check-in*	Gráfico Baseado em Linhas	Variável
C3	Alta	Alta	Baixa	Baixa	Variável	Variável
C4	✗	✗	✗	✗	✗	✓
C5	✗	✗	✗	✗	✓	✓
C6	Ambas	Discreta	Discreta	Discreta	Discreta	Ambas

Tabela 2. Quadro comparativo entre as abordagens pesquisadas e a abordagem proposta.

A abordagem e EvolTrack proporciona ao usuário um método de visualização de todo ciclo de evolução em nível de estrutura de um determinado projeto de software (**C3**). Adicionalmente, vale ressaltar que a abordagem possibilita a utilização de qualquer tipo de fonte de dados (**C1**) e sistema de visualização (**C2**), o que de certa forma pode ser considerado como uma importante contribuição frente às demais propostas pesquisadas.

Outro ponto de contraste com as demais abordagens pesquisadas é que o trabalho proposto possibilita a análise e visualização das informações de projeto sob ambas as perspectivas temporais (**C6**). Em outras palavras, dependendo de como o conector de fonte de dados executa sua política de criação de novos modelos, uma saída com intervalos bem definidos (por exemplo, uma semana) ou uma saída para cada evento ocorrido no projeto pode ser criada.

A utilização do meta-modelo da UML como modelo de dados da abordagem também, pode ser considerado como uma relevante contribuição do trabalho. Visto que

esta representação é de amplo conhecimento e utilização na comunidade de software, sua utilização proporciona uma boa visibilidade do trabalho e aumenta consideravelmente as chances do mesmo ser integrado com outras abordagens.

Um protótipo para a abordagem proposta foi criado. Este, por sua vez, atende a todos os requisitos especificados para atender aos objetivos do trabalho, representando, desta forma, uma bem sucedida prova de conceito da mesma. As seguintes características foram contempladas neste protótipo:

- Visualização da evolução de um projeto na forma de diagramas de classe UML;
- Funcionalidades básicas de navegação entre os diferentes instantes do ciclo de vida do projeto;
- Visualização em modo "vídeo" da história do projeto;
- Total integração com o ambiente de desenvolvimento Eclipse (**C4**);
- Utilização da fonte de dados Lighthouse;
- Utilização do sistema de visualização Eclispe/UML2Tools;
- Atualização automática do histórico (**C5**).

5.2. Limitações

A abordagem proposta não prevê qualquer tipo de mecanismo para proporcionar um recurso de *diff* entre dois instantes da história do projeto. Isto é, um recurso que apresente, dado dois instantes do projeto, o conjunto de elementos que foram criados, removidos e alterados entre estes instantes. Principalmente para sistemas complexos, este recurso facilitaria o entendimento da evolução do projeto.

Em relação ao protótipo implementado, o mesmo não oferece recursos de

marcação de elementos apresentados no diagrama. Esta funcionalidade torna-se útil quando um determinado usuário deseja rastrear todas as evoluções específicas de um determinado elemento do modelo. Adicionalmente, vale citar a incapacidade do protótipo em realizar filtros. Por exemplo, seria de grande interesse a filtragem de projetos. Atualmente, todos os projetos informados pela fonte de dados são apresentados para visualização na ferramenta. A filtragem de elementos do modelo também seria de grande valia em projetos de médio e grande porte, funcionando como uma estratégia para lidar com a explosão de elementos no modelo do projeto.

5.3. Trabalhos Futuros

A atual implementação da abordagem não possui integração com os principais sistemas de controle de versionamento utilizados, como CVS e Subversion. A integração com tais sistemas, através da criação de novos conectores de fonte de dados, possibilitaria a utilização da ferramenta em projetos reais de software, onde boa parte das informações históricas do mesmo se encontra sob a gerência destes sistemas. Entretanto, é importante frisar que este tipo de integração deverá ser acompanhado por algum método de engenharia reversa de forma que os elementos estruturais possam ser resgatados do código fonte sob versionamento.

Outro trabalho de grande interesse seria a criação de um conector de fonte de dados para o sistema Odyssey-VCS 2, onde modelos já em formato UML são mantidos sob versionamento. Desta forma, a transição entre o formato externo para o formato interno seria praticamente direta. Além disto, esta integração provê um meio de visualização para modelos armazenados neste sistema. Algumas iniciativas, como a criação de modelos marcados com determinadas métricas de projeto (PRUDENCIO et

al., 2007) e posterior armazenamento no Odyssey-VCS2, se beneficiariam com esta integração, uma vez que ganhariam, sem qualquer esforço adicional, um método de visualização. A mesma idéia pode ser aplicada para qualquer trabalho que tenha como saída modelos UML e que estes sejam armazenados no Odyssey-VCS2.

Referências Bibliográficas

(AHO et al., 2006) AHO, A.V., SETHI, R., ULLMAN, J.D., 2006, "A Simple Syntax-Directed Translator", In: *Compilers: Principles, Techniques and Tools*, 2ª Edição, Capítulo 2, MA, EUA, Addison-Wesley.

(BALL e EICK, 1996) BALL, A.T, EICK, S.G., 1996, "Software Visualization in the Large", *IEEE Computer*, *vol. 29*, *n. 4*, pp. 33-43, Abril.

(BUGZILLA, 2008) Bugzilla, 2008, Em: www.bugzilla.org, Acesso em Maio.

(CARROL, 2001) CARROL, J., 2001, "Jesus, a Jew?", In: Houghton Mifflin Books (ed), *Constantine's Sword: The Church and the Jews – A History*, Capítulo 9, EUA, Mariner Books.

(CEDERQVIST et al., 2006) CEDERQVIST, P. *et al.*, 2006, *Version Management with CVS*, 1ª Ed., EUA, Network Theory Ltd .

(CLAYBERG e RUBEL, 2006) CLAYBERG, E., RUBEL, D., 2006, "Eclipse Infrastructure", In: Gamma, E., Nackman, L., Wiegand, J. (eds), *Eclipse: Building Commercial-Quality Plug-ins*, 2ª Edição, Cap. 3, Boston, MA, EUA, Addison Wesley Professional.

(COLLBERG et al., 2003) COLLBERG, C., KOBOUROV, S., NAGRA, J., et al., 2003, "A System for Graph-Based Visualization of the Evolution of Software", *In: Proceedings of the 2003 ACM Symposium on Software Visualization,* ACM Press, pp. 957-968, San Diego, CA, EUA, Junho.

(CRNKOVIC et al., 2004) CRNKOVIC, I., STAFFORD, J., SCHMIDT, H., WALLNAU, K., 2004, "Component-Based Software Engineering", 7th International Symposium, CBSE, Edinburgh, UK, Maio, Lecture Notes in

Computer Science 3054, Springer.

(DA SILVA et al., 2006) DA SILVA, I. A., CHEN, P. H., VAN DER WESTHUIZEN, C., RIPLEY, R. M., VAN DER HOEK, A., 2006, "Lighthouse: coordination through emerging design", *In: Proceedings of the 2006 OOPSLA Workshop on Eclipse Technology Exchange (Portland, Oregon, October 22 - 23, 2006). eclipse '06. ACM Press*, New York, NY, pp. 11-15.

(EICK et al., 1992) EICK, S., STEFFEN, J., SUMMER, E., 1992, "Seesoft: A Tool for Visualizing Line-Oriented Software Statistics", *IEEE Transactions on Software Engineering, vol. 18, n. 11,* pp. 957-968, Novembro.

(ECLIPSE, 2008) ECLIPSE FOUNDATION, 2008, Em: http://eclipse.org, Acesso em Maio.

(ESTUBLIER, 2000) ESTUBLIER, J., 2000, "Software Configuration Management: A roadmap", *In: Proceedings of the 22th Conference on The Future of Software Engineering, International Conference on Software Engineering (ICSE)*, pp. 279-289, Limerick, Irlanda.

(FROEHLICH e DOURISH, 2004) FROEHLICH, J., DOURISH, P., 2004, "Unifying Artifacts and Activities in a Visual Tool for Distributed Software Development Teams", *In: Proceedings of the 26th International Conference on Software Engineering (ICSE)*, IEEE Press, pp. 387-396, Edinburgh, Escócia, Maio.

(GAMMA et al, 1994) GAMMA, E., HELM, R., JOHSON, R., VLISSIDES, J., 1994, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley.

(GERMAN et al., 2006) GERMAN, D.M., HINDLE, A., JORDAN, N., 2006, "Visualizing the Evolution of Software using SoftChange", *International Journal*

of Software Engineering and Knowledge (IJSEKE), vol. 16, n. 1, pp. 5-21, Fevereiro.

(HARRISON et al., 2000) HARRISON, W., OSSHER, H., TARR, P., 2000, "Software Engineering Tools and Enviroments: A Roadmap", *In: Proceedings of the 22th Conference on The Future of Software Engineering, International Conference on Software Engineering (ICSE)*, pp. 261-277, Limerick, Irlanda.

(HOLT e MANCORIDIS, 1994) HOLT, R.C., MANCORIDIS, S., 1994, "A Framework for Specifying and Visualizing Architectural Design", *Relatório Técnico CSRI-300*, Computer Science Research Institute, Universidade de Toronto, Junho.

(HOLT e PAK, 1996) HOLT, R.C., PAK, J., 1996, "GASE: Visualizing Software Evolution-in-the-Large", In: *Proceedings of the Working Conference on Reverse Engineering (WCRE'96)*, pp. 163-167, Monterey, CA, EUA, Novembro.

(HUNT e MCILROY, 1976) HUNT, W.J., MCILROY, D.M., 1976, "An algorithm for differential file comparison", In: *Relatório Técnico 41*, AT&T Bell Laboratories, Inc., Murray Hill, NJ, EUA.

(HUNT e SZYMANSKI, 1977) HUNT, W.J., SZYMANSKI, G.T., 1977, "A fast algorithm for computing longest common subsequences", *Communications of the ACM*, v. 20, n. 5, pp. 350-353, Maio.

(ISO, 2005) Norma ISO/IEC 25000 (SQuaRE), 2005, "Software product Quality Requirements and Evaluation – Guide to SQuaRE", Em: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35683, Acesso em Maio.

(KERNIGHAN e RITCHIE, 1978), KERNIGHAN, B., RITCHIE, D., 1978, *The C Programming language*, 1ª Ed., Englewood Cliffs, NJ, EUA, Prentice-Hall.

(MURTA et al., 2008) MURTA, L., CORREA, C., PRUDENCIO, J.G., WERNER, C., 2008, "Towards odyssey-VCS 2: improvements over a UML-based version controle system", *Proceedings of the 2008 international workshop on Comparison and versioning of software models***,** pp. 25-30, Leipzig, Alemanha, Maio.

(OMG, 2008) Unified Modeling Language (UML), 2008, Em: http://www.omg.org/uml, Acesso em Maio.

(PRESSMAN, 2006) PRESSMAN, R.S., 2006, "Software e Engenharia de Software", In: *Engenharia de Software*, 6ª Edição, Capítulo 1, McGraw-Hill.

(PRUDENCIO et al., 2007) PRUDENCIO, J. G. G., MURTA, L. G. P., WERNER, C. M. L., "Políticas de Controle de Concorrência no Desenvolvimento Distribuído de Software", In: *Workshop de Desenvolvimento Distribuído de Software (WDDS)*, pp. 57-64, João Pessoa, 2007.

(RATZINGER et al., 2005) RATZINGER, J., FISCHER, M., GALL, H., 2005, "EvoLens: Lens-View Visualizations of Evolution Data", In: *Proceedings of International Workshop on Principles of Software Evolution* (*IWPSE'05)*, pp. 103-112, Lisboa, Portugal, Setembro.

(ROBERTSON et al., 1993) ROBERTSON, G., CARD, S., MACKINLAY, J., 1993, "Information visualization using 3-D interactive animation", *Communications of the ACM*, vol. 36, n. 4, pp. 57-71, Abril.

(SARKAR e BROWN, 1992) SARKAR, M., BROWN, H.M., 1992, "Graphical fisheye views of graphs", In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 83-91, Monterey, CA, EUA, Maio.

(SUN MICROSYSTEMS, 2008) SUN MICROSYSTEMS, INC., 2008, Em: http://java.sun.com/, Acesso em Maio.

(SUSSMAN et al., 2004) SUSSMAN, B.C., FITZPATRICK, B.W., PILATO, C.M., 2004, *Version Control with Subversion*, 1ª Ed., Stanford, CA, EUA, O'Reilly.

(SULLIVAN et al., 2001) SULLIVAN, K., GRISWOLD, W., CAI, Y., et al., 2001, "The structure and value of modularity in software design", ACM SIGSOFT Software Engineering Notes, pp. 99-108, Setembro.

(UML2, 2008) Projeto Eclipse UML2, 2008, Em: http://www.eclipse.org/uml2, Acesso em Maio.

(UML2Tools, 2008) Projeto Eclipse UML2Tools, 2008, Em: http://www.eclipse.org/modeling/mdt, Acesso em Maio.

(VOINEA et al., 2005) VOINEA, L., TELEA, A., WIJK, J.J., 2005, "CVSscan: Visualization of Code Evolution", *In: Proceedings of the 2005 ACM Symposium on Software Visualization*, pp. 47-56, St. Louis, Missouri, EUA, Maio.

(VOINEA e TELEA, 2006) VOINEA, L., TELEA, A., 2006, "CVSGrab: Mining the History of Large Software Projects", *In: Proceedings of Eurographics/IEEE-VGTC Symposium on Visualization. IEEE Press,* pp. 187-194, Lisboa, Portugal, Maio.

(WALL et al., 2000) WALL, L., CHRISTIANSEN, T., ORWANT, J., 2000, *Programming Perl*, 3ª Ed., Stanford, CA, EUA, O'Reilly.

(WENZEL, 2005) WENZEL, S., "Automatic detection of incomplete instances of structural patterns in UML class diagrams", Nordic Journal of Computing, vol. 12, no. 14, pp. 379-394.