Visualizando a Evolução de Software no Desenvolvimento Distribuído

Rafael Cepêda[1], Leonardo Murta[2], Cláudia Werner[1]

[1] Programa de Engenharia de Sistemas e Computação – COPPE/UFRJ
Caixa Postal 68.511, Rio de Janeiro, RJ, 21945-970

{rcepeda, werner}@cos.ufrj.br

[2] Instituto de Computação – Universidade Federal Fluminense
Rua Passo da Pátria 156, Niterói, RJ, 24210-240

leomurta@ic.uff.br

Resumo. *Desenvolver software de forma distribuída tem se tornado crescentemente um paradigma real de desenvolvimento nas organizações de Tecnologia da Informação no mundo. Porém, junto com seus diversos aspectos positivos, encontram-se diferentes barreiras que devem ser superadas para que o processo de desenvolvimento produza projetos de sucesso ao longo do tempo. Este artigo apresenta um mecanismo de visualização da evolução de software capaz de ser integrado em um processo de desenvolvimento distribuído e com isso apresenta uma ferramenta por onde desenvolvedores espalhados geograficamente podem melhor entender a evolução de seus projetos e identificar, através de uma visão global do projeto, diferentes características relevantes do mesmo.*

Palavras-Chave: Visualização de Software, Evolução de Software, Desenvolvimento Distribuído.

Visualizing Software Evolution in Distributed Delevelopment

Abstract. *Developing software in a distributed setting is increasingly becoming a real development paradigm for Information Technology organizations around the world. However, together with all its good aspects, it raises different barriers that must be overcome in order to achieve success. This paper presents a mechanism for software evolution visualization capable of being fully integrated in a distributed development process and therefore presents a tool where geographically distributed developers can better understand the evolution of their projects and identify, through a global vision of the project, its various relevant features.*

Keywords: Software Visualization, Software Evolution, Distributed Development.

(Received July 11, 2008 / Accepted October 20, 2008)

1. Introdução

O desenvolvimento distribuído de software (DDS) cada vez mais se torna uma realidade nas grandes organizações de Tecnologia da Informação dominantes do mercado [12]. Dentre os diferentes aspectos que tornam isto uma realidade, estão os menores custos no desenvolvimento de software em determinados países, aquisições de novas empresas estrangeiras e outros [13]. Neste contexto, diferentes fatores representam grandes barreiras quando se trata de desenvolvimento de software. Fatores como dificuldade de comunicação, coordenação, gerenciamento e a dificuldade na identificação de pessoas necessárias para uma comunicação mais eficaz são apenas uma pequena amostra dos problemas enfrentados neste cenário [11].

Além disto, devido à natureza intangível do software, alinhado com seu crescente grau de complexidade, as atividades de desenvolvimento tornam-se tarefas de difícil gerenciamento e execução [1]. Logo, 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 [17]. 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 das atividades de desenvolvimento pode se tornar uma tarefa de menor custo e complexidade.

Este artigo apresenta um mecanismo capaz de resgatar e visualizar o ciclo de evoluções de um projeto de software sendo desenvolvido de forma distribuída. Com isso, o artigo evidencia que algumas dificuldades, apresentadas na seção 2, criadas ou reforçadas pelo desenvolvimento distribuído de software podem ser amenizadas com a adoção desta abordagem.

Seguindo esta introdução, o restante do artigo está organizado da seguinte forma: a seção 2 apresenta o cenário de desenvolvimento distribuído que será utilizado ao longo do artigo; a seção 3 apresenta a abordagem EvolTrack, utilizada para a visualização da evolução de projetos de software; a seção 4 apresenta como a abordagem anteriormente descrita pode ser utilizada no cenário descrito na seção 2. Além disto, essa seção apresenta, através de um breve exemplo, como algumas barreiras impostas por projetos desta natureza podem ser amenizadas com a utilização do EvolTrack; a seção 5 apresenta os trabalhos relacionados; e, a seção 6 apresenta as conclusões deste trabalho.

2. Cenário de DDS sem EvolTrack

Suponha, sem perda de generalidade, um cenário de desenvolvimento de software onde existem três desenvolvedores: *desenvolvedor 1*, *2* e *3*. Adicionalmente, vale ressaltar que, neste cenário, cada desenvolvedor citado encontra-se em um local geograficamente distinto dos demais. Todos os três fazem parte de uma única equipe de desenvolvimento, com a finalidade de construir um sistema para uma determinada loja. Além disto, o projeto conta com um repositório de código fonte central onde, através de um sistema de versionamento, os desenvolvedores armazenam e gerenciam seus produtos de trabalho.

Tipicamente, a dinâmica deste desenvolvimento distribuído envolveria previamente uma distribuição de tarefas a serem alocadas para cada desenvolvedor disponível na equipe. À medida que atividades vão sendo completadas pelos desenvolvedores, o software evolui e atinge mais uma etapa do seu desenvolvimento. Este ciclo de evoluções é, então, repetido até que todas as atividades tenham sido completadas e, conseqüentemente, a fase de desenvolvimento tenha chegado ao seu fim.

Neste contexto, algumas dificuldades, que podem ser observadas tanto pelos desenvolvedores quanto pela pessoa responsável por sua coordenação, são expostas:
P1. Como um desenvolvedor, com sua visão local do projeto, consegue observar o atual estágio global (i.e. a soma dos produtos parciais, ainda em construção, de todos desenvolvedores) do desenvolvimento do mesmo? **P2.** Como o coordenador consegue identificar o grau de evolução do desenvolvimento como um todo? **P3.** Como um desenvolvedor ou coordenador consegue visualizar se o desenvolvimento está convergindo ou divergindo de um possível modelo de projeto construído inicialmente? **P4.** Como um desenvolvedor consegue identificar, ou ao menos inferir, o responsável pelo desenvolvimento de uma parte do software?

O desenvolvimento distribuído de software, devido a sua natureza, dificulta muitas vezes que as questões apresentadas anteriormente sejam satisfatoriamente respondidas. A seção 4 retomará o exemplo aqui descrito, acrescentando o elemento EvolTrack neste cenário e evidenciando como a utilização do mesmo pode apoiar na resposta a estas questões.

3. Abordagem EvolTrack

A abordagem EvolTrack, apresentada abstratamente na Figura 1, proporciona uma infra-estrutura capaz de capturar o ciclo de evolução de um projeto de software e, através da visualização, apresentar de forma temporal este ciclo de evolução previamente identificado. Repare que, além do próprio EvolTrack, existem dois elementos que integram esta visão do mecanismo: Fonte de Dados e Visualizadores. O primeiro, diz respeito a qualquer sistema externo capaz de armazenar e prover informações sobre o histórico de evoluções de um determinado projeto. Neste contexto, podem ser citados como exemplos de fontes de dados sistemas de versionamento de código fonte de projetos de software, como CVS [4] e Subversion [18], sistemas de versionamento de modelos do projeto de software, como Odyssey-VCS [14], ou até mesmo ambientes de desenvolvimento de software como Eclipse.

Desta forma, as informações capturadas a partir de uma dada fonte de dados servirão como base para a criação de toda uma representação do ciclo de evoluções do projeto de software por parte do EvolTrack. Portanto, para viabilizar este esquema de representação, uma implementação do meta-modelo da UML foi adotada na construção do mecanismo, visando refletir um dado instante do projeto. Assim, o ciclo de evolução do projeto é representado internamente como uma seqüência de modelos descritos por este meta-modelo. Em um cenário típico de utilização do mecanismo, pode ser observado que, à medida que o mesmo detecta (ou é informado a partir da fonte de dados) que o projeto sob análise sofreu uma evolução, um novo modelo será criado de forma a representar a nova configuração para este projeto. Este novo modelo será, então, enviado para o elemento Visualizador para a apresentação visual desta evolução.

Assim, um Visualizador pode ser entendido como um segundo sistema externo capaz de apresentar visualmente as informações do projeto fornecidas pela abordagem EvolTrack. É importante ressaltar que a forma de representação visual adotada pelo visualizador poderá seguir diferentes estilos. Um exemplo de representação visual que poderia ser utilizada neste contexto seria a de diagramas de classe UML. Porém, qualquer outro tipo de notação visual que consiga representar de maneira satisfatória as informações fornecidas pelo mecanismo também poderia ser adotado.

A Figura 2 apresenta como o mecanismo foi dividido em componentes de software que, ao cooperarem entre si, proporcionam a funcionalidade anunciada. Um componente Conector de Fonte de Dados é responsável pela comunicação com as Fontes de Dados externas. Suas responsabilidades incluem resgatar as informações de projeto e converter estas informações da representação externa (e.g., código fonte) para a representação interna (i.e., modelos UML).

Atualmente, dois tipos de conectores de fonte de dados podem ser instanciados, a saber: Conector de Fonte de Dados Simples e Conector de Fonte de Dados Complexo. O primeiro encapsula toda a lógica de extração e transformação dos dados oriundos de uma determinada fonte de dados. Neste caso, tudo que a infra-estrutura criada pela abordagem EvolTrack precisa saber é como inicializar este conector, uma vez que todo o restante do trabalho, incluindo a periodicidade da extração, é definida pelo próprio criador do conector.

Em contrapartida, a infra-estrutura permite que desenvolvedores criem os chamados Conectores de Fonte de Dados Complexos. A idéia por trás deste tipo de conector é permitir que um número variável e, a priori, indefinido de transformações possa ser executado sobre os dados extraídos inicialmente. Desta forma, este tipo de conector é composto por duas unidades essenciais. A primeira, chamada de Extrator, é a parte do conector responsável por realizar as interações com a fonte de dados, extraindo e disponibilizando seus dados para posterior transformação. A segunda parte é composta pelos Transformadores. Estes formam uma cadeia de transformações que deveram ser aplicadas sobre os dados extraídos pelo Extrator. A Figura 3 ilustra esta estrutura.

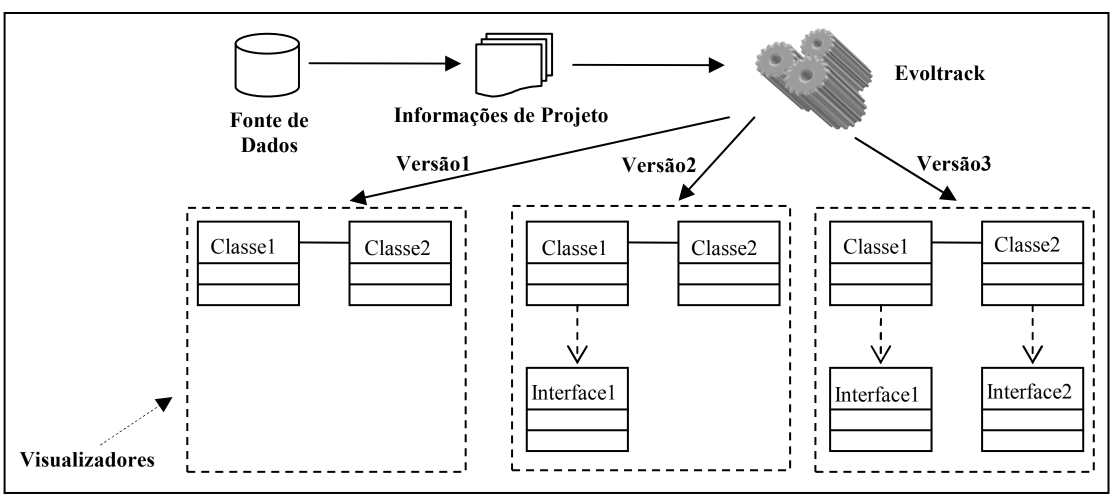

Figura 1. Uma visão geral da abordagem EvolTrack.

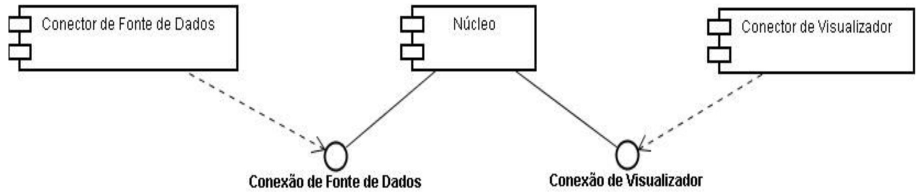

Figura 2. Componentes da abordagem EvolTrack.

Assim, através da interface fornecida pelo EvolTrack, pode-se optar pela utilização de um conector ao invés de outro ou até mesmo compor uma cadeia de transformações com um dado conector e seus transformadores instalados. Portanto, é importante ressaltar que uma das principais diferenças entre um conector de fonte de dados simples e um conector de fonte de dados complexo está na divisão e execução de suas partes. O primeiro é visto pela infra-estrutura EvolTrack apenas como se tivesse uma única parte, e o segundo é explicitamente dividido em duas partes, extrator e transformador. O controle da execução do segundo é feito pela infra-estrura, isto é, é ela quem agenda a execução de um extrator e depois de seus transformadores em cadeia. Já o primeiro é apenas inicializado pela infra-estrutura. O modo como ocorrerá suas extrações e transformações internas é de responsabilidade do próprio conector.

Ainda na Figura 2, 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 (i.e., modelos UML) para o formato específico de saída (e.g., diagrama de classes UML).

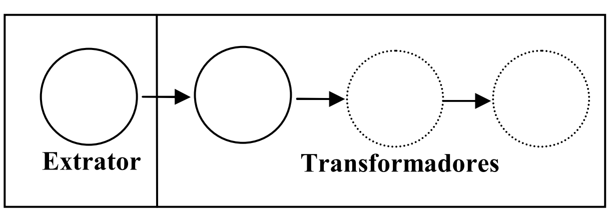

Figura 3. Conector de Fonte de Dados Complexo.

Como será observado posteriormente, ao longo deste artigo, no presente momento a infra-estrutura EvolTrack possui disponível um único Conector de Visualizador capaz de representar os modelos UML, gerados anteriormente, em diagramas de classe UML juntamente com algumas informações de controle adicionais, como o responsável por uma dada evolução no modelo e a data de ocorrência desta evolução.

Adicionalmente, vale ressaltar que este conector ainda possibilita a visualização de métricas associadas a um dado modelo, onde cada métrica deve estar codificada (ou marcada) na forma de valores etiquetados (do inglês, *tagged values*) dentro do modelo UML gerado inicialmente.

Note que, esta apresentação ocorre na forma de escalas de cores [7], onde cada métrica, que pode aparecer em um dado modelo está associada a uma escala. Esta escolha de escala é feita pelo usuário do EvolTrack quando o mesmo informa à ferramenta o perfil UML que será utilizado durante a visualização. Este perfil contém todos os elementos adicionais aos elementos UML padrão que um dado modelo pode apresentar. Vale ressaltar que perfil, estereótipos e valores etiquetados constituem o mecanismo padrão de extensão da UML [2].

Com isso, uma vez que o conector de fonte de dados gere um modelo UML contendo métricas (e.g., grau de acoplamento de uma classe) que representem características de um dado projeto, a abordagem EvolTrack será capaz de proporcionar um meio para o acompanhamento visual da evolução de tais características. É o que acontece, por exemplo, com a integração entre o EvolTrack e o projeto Orion [16].

O projeto Orion visa, a partir do histórico de controle de versões de um determinado projeto, sugerir para cada elemento do seu modelo se uma política de gerência de configuração otimista ou pessimista deve ser aplicada ao mesmo. Para isto, algumas métricas referentes ao projeto são calculadas e disponibilizadas no modelo do projeto que fora utilizado como entrada, bem como o tipo de política que deverá ser adotada para um elemento.

Com a criação de um Conector de Fonte de Dados Complexo contendo um extrator para o sistema Odyssey-VCS, foi possível obter todas as informações necessárias para a execução dos cálculos supracitados. Então, de forma a possibilitar que o mesmo conector fosse utilizado com ou sem as regras aplicadas pelo Orion, criou-se um transformador para este projeto. Assim, caso o usuário queira utilizar o conector para Odyssey-VCS com o Orion, basta instalar o transformador e adicioná-lo, através da interface, à cadeia de transformação do mesmo. Caso contrário, basta retirá-lo da cadeia.

Então, a integração do EvolTrack com o projeto Orion possibilitou a visualização da evolução das métricas geradas pelo mesmo. Em outras palavras, possibilitou o acompanhamento visual de como o grau de concorrência de desenvolvimento de um dado elemento evoluiu ao longo do tempo no contexto específico de um projeto. Note que este sistema de visualização e controle de métricas pode ser utilizado no contexto de qualquer projeto, bastando apenas que o conector de fonte de dados utilizado gere como saída um modelo UML contendo as marcações pertinentes e que o usuário informe previamente à ferramenta o tipo de perfil UML que será utilizado para interpretar tais modelos.

Por fim, o componente Núcleo é responsável por manter os modelos criados para cada projeto através do Conector de Fonte de Dados e, adicionalmente, notificar todos os interessados em receber atualizações sobre a evolução de um dado projeto. Neste caso, todo Conector de Visualizador que for previamente cadastrado no componente Núcleo como interessado será notificado. Com isso, o usuário, através da interface visualizadora, poderá acompanhar em tempo real, por exemplo, a evolução do projeto em construção.

Assim, a instanciação destes três componentes concretiza o mecanismo em um sistema de software. Portanto, pode ser observado que o mesmo, a principio, não está atrelado a nenhum tipo de fonte de dados específica e a nenhum tipo de sistema de visualização específico, apesar de ser dependente da infra-estrutura Eclipse. Por exemplo, o EvolTrack poderia ser utilizado para resgatar as informações de um projeto oriundas do sistema Subversion e, posteriormente, apresentar a evolução do mesmo de forma diagramática no ambiente Eclipse. Desta forma, um Conector de Fonte de Dados para o Subversion deveria ser criado, bem como um Conector de Visualizador que consiga realizar uma representação diagramática no Eclipse. Se, em contrapartida, deseja-se capturar as informações de projeto a partir de um sistema de versionamento de modelo como o Odyssey-VCS, um Conector de Fonte de Dados específico para este sistema deveria ser criado.

4. Cenário de DDS com EvolTrack

Retomando o exemplo descrito na seção 2, a Figura 4 ilustra este cenário de utilização, contemplando também a utilização da abordagem EvolTrack**.** Cada desenvolvedor conta com seu ambiente de desenvolvimento de software local que, neste caso, representa uma instância da plataforma Eclipse, criando, assim, três diferentes espaços de trabalho $\mathbf{W_1}$**,** $\mathbf{W_2}$ **e** $\mathbf{W_3}$. Neste ponto, entra em ação o projeto Lighthouse [6], que, por sua vez, representa um *plug-in* da plataforma Eclipse capaz de capturar eventos de desenvolvimento gerados na plataforma e popular um banco de dados central com estes eventos. Os eventos capturados são, em geral, apenas aqueles relacionados com a atividade de codificação, como, por exemplo, a criação, remoção e modificação de classes, métodos, atributos e pacotes. A melhor utilização da ferramenta ocorre quando o desenvolvedor utiliza dois monitores, onde em um primeiro monitor aparece todo seu ambiente de programação usual e em um segundo monitor aparece os resultados gerados pela utilização do plug-in EvolTrack.

Neste momento, é possível observar que a simples utilização do banco de dados Lighthouse como fonte de dados para a abordagem EvolTrack possibilita a este o acesso simultâneo ao conjunto de modificações ou evoluções geradas pelos três diferentes desenvolvedores do projeto. Com base nestas informações, o EvolTrack apresentará posteriormente na plataforma Eclipse, aos mesmos desenvolvedores, uma visão integrada de todo o projeto e, além disto, permitirá que cada um destes possa navegar ao longo do tempo e visualizar os diferentes estados ou configurações pelos quais o projeto em desenvolvimento passou até chegar ao estado atual de sua estrutura. Note que, neste caso, este estado será um reflexo do somatório das estruturas encontradas em cada espaço de trabalho $\mathbf{W_i}$ pertencente a cada desenvolvedor.

Por exemplo, se inicialmente o desenvolvedor 1 cria a classe *Loja*, em seguida o desenvolvedor 2 cria a classe *Produto* e, por fim, o desenvolvedor 3 cria a classe *Cliente*, o EvolTrack estará ao final apresentando aos três desenvolvedores, simultaneamente, um diagrama de classes UML contendo as três classes criadas em separado, possibilitando, desta forma, um melhor entendimento global do andamento e da estrutura do projeto em questão (**P1**). Note que, esta noção de entendimento, embora seja intuitiva, em geral não é facilmente alcançada, principalmente em projetos com equipes distribuídas, onde o fator comunicação pode ser comprometido pela distância das equipes. Porém, neste caso, além dos três desenvolvedores estarem com a mesma noção de entendimento do projeto, caso um dos dois primeiros desenvolvedores quisesse realizar alguma modificação que possivelmente poderia gerar impactos em uma implementação da classe *Cliente*, ele saberia de imediato a quem recorrer para efetuar possíveis esclarecimentos sobre esta classe, antes mesmo desta estar disponível no repositório do projeto (**P4**).

Complementando o cenário descrito anteriormente, deve ser observado que a utilização do EvolTrack é concretizada, nesta configuração, via utilização dos seguintes *plug-ins* para Eclipse: **EvolTrack-LH**, **EvolTrack-Kernel** e **EvolTrack-Eclipse**. O primeiro *plug-in* representa o Conector de Fonte de Dados que é capaz de interagir com o sistema Lighthouse, extraindo as informações pertinentes e transformando-as em informações de projeto na representação interna, isto é, modelos UML. Assim, o usuário deve selecionar como fonte de dados para este projeto o sistema Lighthouse, conforme sugere a Figura 5 O segundo *plug-in* implementa o componente Núcleo discutido anteriormente. O terceiro *plug-in* representa o Conector de Visualizador que, neste caso, é responsável por, a partir dos modelos UML fornecidos pelo **EvolTrack-Kernel**, apresentar na forma de diagramas de classe UML os diversos estados do projeto.

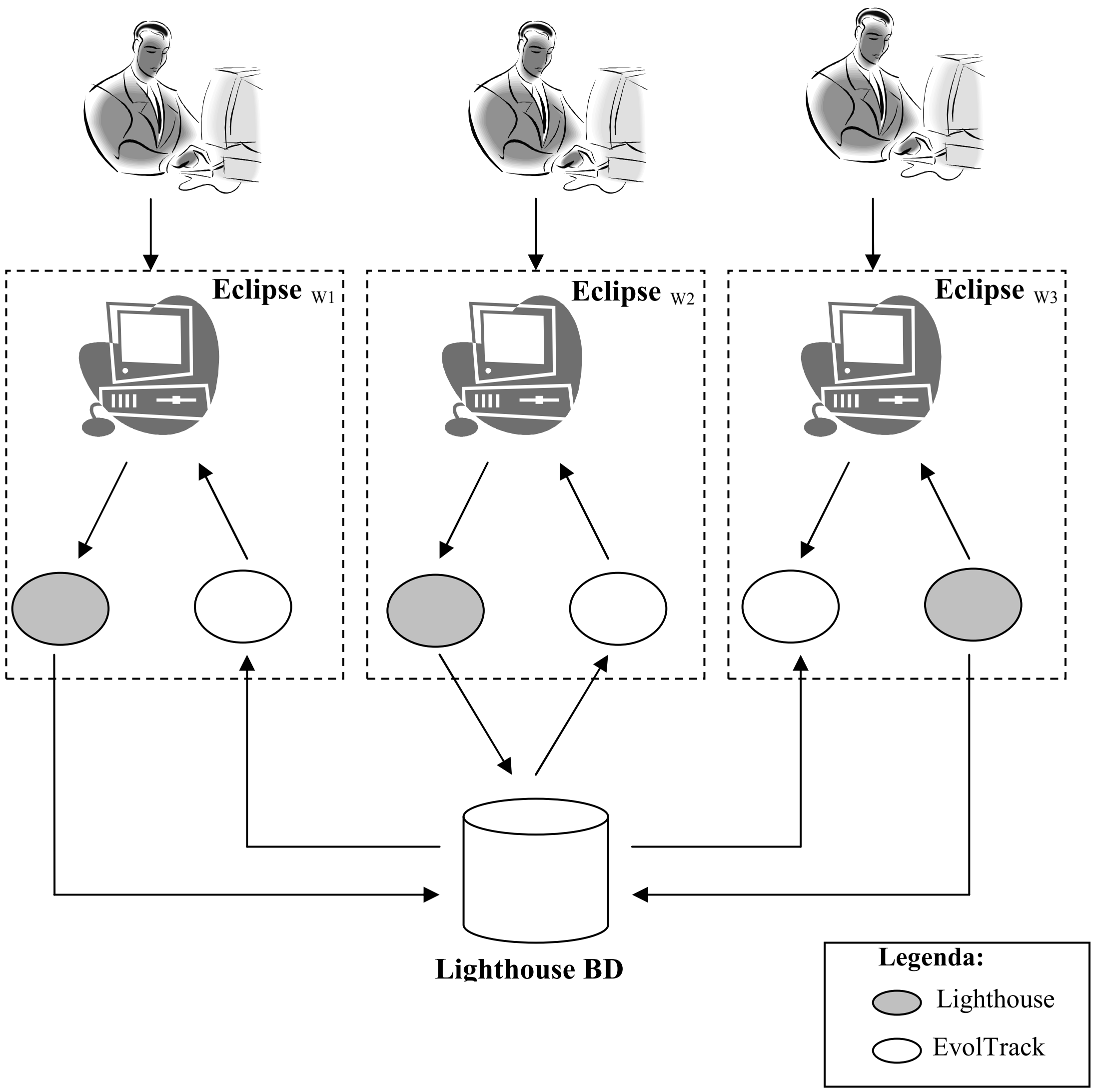

Figura 4. Cenário de utilização do EvolTrack.

Além disto, este *plug-in* fornece uma interface com usuário que, além de apresentar estes modelos, também possibilita uma navegação pela história deste projeto, tanto de forma manual como de forma automática, analogamente a um vídeo.

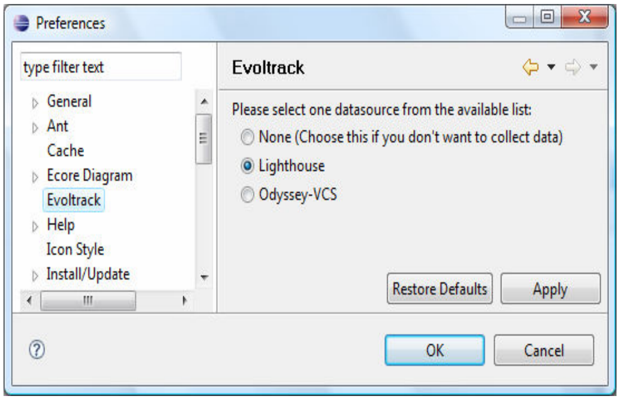

Figura 5. Seleção da fonte de dados.

A Figura 6 e Figura 7 apresentam a interface fornecida pelo **EvolTrack-Eclipse,** juntamente com a visualização do cenário descrito anteriormente (Figura 6), onde três desenvolvedores geograficamente distribuídos criaram as seguintes classes: *Loja*, *Produto* e *Cliente*.

Note que no instante representado pela Figura 6 o último desenvolvedor a contribuir com o projeto foi o desenvolvedor 3, cuja contribuição resultou na criação da classe *Cliente*. Posteriormente, pode ser observado (vide Figura 7) que o desenvolvedor 1 criou um atributo do tipo *Cliente* na classe *Loja*, o que, por sua vez, resultou em uma evolução estrutural do projeto representado pelo aparecimento de uma associação entre ambas as classes. Neste exemplo, o desenvolvedor 1, ao notar a criação da classe *Cliente* pelo desenvolvedor 3 no projeto, através do aparecimento de uma nova versão do modelo em seu visor executando o EvolTrack, rapidamente fez uma solicitação para que o desenvolvedor 3 disponibilizasse a mesma via sistema de versionamento, ganhando assim maior agilidade na elaboração de sua atividade referente à classe *Loja*.

Adicionalmente, é importante notar que na Figura 8 o sistema de visualização e controle de métricas, apresentado brevemente na seção 3, encontra-se habilitado. Neste exemplo, temos uma única possível métrica associada a modelo, a saber, grau de acoplamento. Esta métrica, neste caso, representa um número inteiro que pode variar de 1 a 100, onde quanto maior o número, maior o grau de acomplamento. Além disto, uma escala de cor do tipo LOCS [15] foi associada à esta métrica, conforme pode ser observado na figura.

Desta forma, pode ser observado que as classes *Cliente* e *Loja* possuem, neste instante do tempo, o mesmo grau de acomplamento, seguindo as regras de medição do Conector de Fonte de Dados utilizado neste cenário. E, além disto, a classe *Produto* possui um grau de acomplamente menor do que as demais, uma vez que sua coloração de borda está situada, ao longo da escala de cores escolhida, em uma posição anterior a coloração das demais classes do modelo.

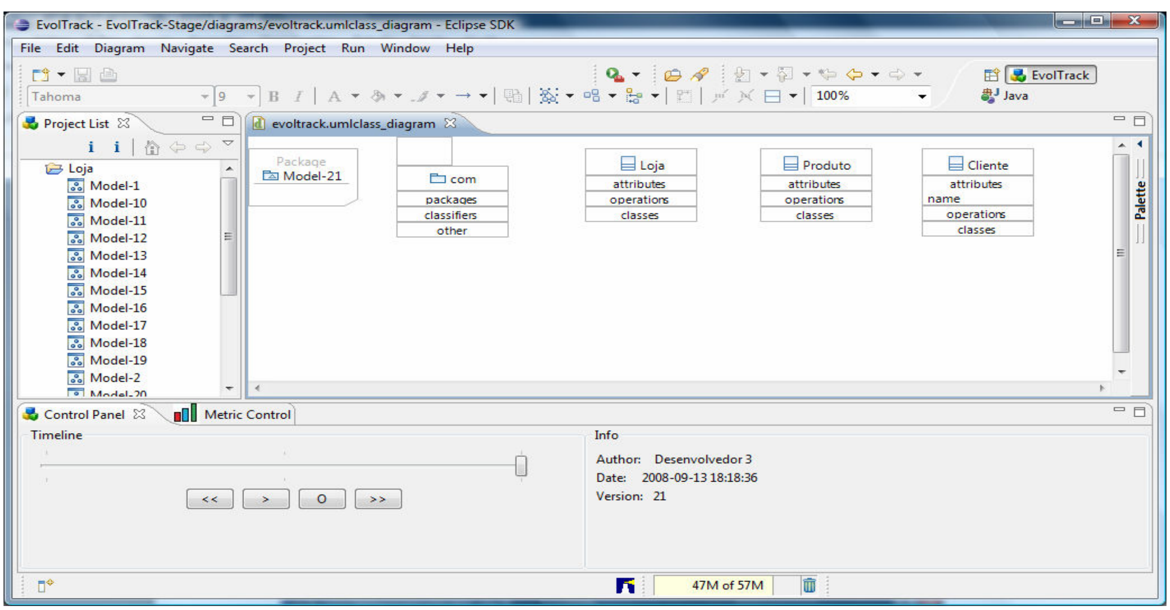

Figura 6. Criação da classe *Cliente* pelo Desenvolvedor 3.

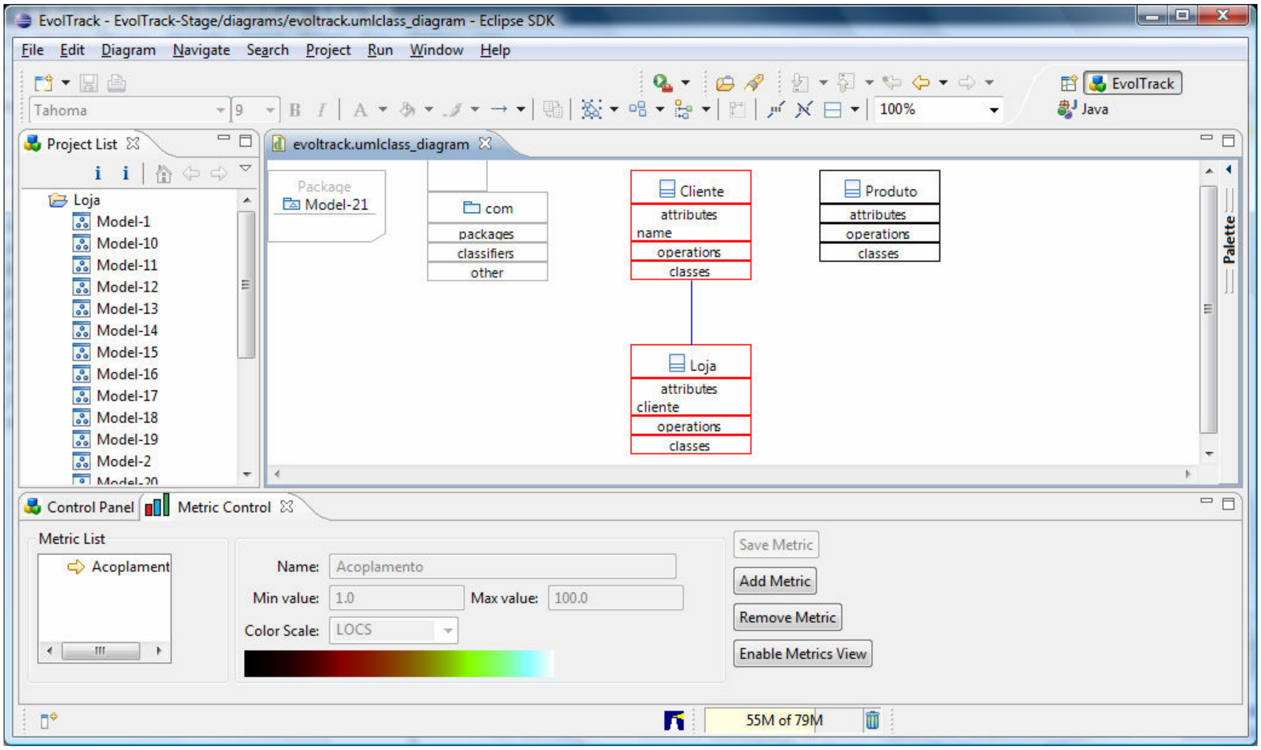

Figura 7. Criação pelo Desenvolvedor 1 de uma relação de associação entre a classe *Loja* e a classe *Cliente*.

Note que, a possível utilização das mesmas ferramentas pelo coordenador possibilitaria a este uma boa noção do grau de evolução do projeto, o que permitiria tomar as devidas ações caso algo estivesse fora do comum (**P2**). Por exemplo, navegando através da barra do tempo (*Timeline* na Figura 6), caso o coordenador constate a ausência de evoluções de um determinado desenvolvedor, possivelmente indicaria que este desenvolvedor não está trabalhando em sua atividade ou que sua comunicação com a rede tenha sido interrompida. Note que a utilização de um grande monitor, neste caso pelo coordenador, aumentaria ainda mais esta percepção do projeto.

Além disto, tendo em mãos o modelo do projeto especificado em uma etapa anterior do processo, a simples comparação visual deste com os sucessivos diagramas apresentados pelo EvolTrack, mesmo que de forma manual, possibilitaria a identificação de possíveis divergências entre o que fora especificado e o que está sendo construído pela equipe distribuída (**P3**). Vale notar que a própria ferramenta também auxiliaria na identificação do desenvolvedor responsável por esta divergência.

5. Trabalhos Relacionados

A ferramenta Augur [9] utiliza uma abordagem de visualização baseada em linhas de código fonte [8]. 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.. Porém, a falta de integração desta ferramenta com um ambiente de desenvolvimento de software faz com que a utilização do EvolTrack, ferramenta já integrada ao Eclipse, proporcione menos sobrecarga ao desenvolvedor durante a fase de construção. Além disto, a possibilidade de identificação da convergência ou divergência do desenvolvimento para um modelo de projeto já construído é extremamente prejudicada em função do tipo de representação visual adotada, baseada em linhas de código fonte, conforme mencionado anteriormente.

Já German et al. [10] definem 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 ferramenta softChange [10] foi projetada com o intuito de, 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. Entretanto, ao contrário da abordagem EvolTrack que possibilita um acompanhamento em tempo real do desenvolvimento do projeto, a ferramenta softChange apenas conseguirá fornecer as informações sobre projeto depois destas serem armazenadas no repositório CVS, o que limita sua aplicação.

Além destas ferramentas, outras abordagens [5] [3], que não se concentram em técnicas de visualização, também tratam de alguns problemas discutidos previamente neste artigo. Vale destacar que estas ferramentas disponibilizam diferentes tipos de funcionalidades relacionadas a colaboração entre equipes de desenvolvimento geograficamente distribuídas. Em [5], por exemplo, funcionalidades de *chat* e compartilhamento de tela são disponibilizados aos usuários. Já em [3], uma funcionalidade de edição simultânea de modelos entre diversos desenvolvedores pode ser encontrada. Entretanto, ambas as ferramentas não possuem a capacidade de acompanhamento em tempo real da construção do software, que a abordagem EvolTrack provê. Neste contexto, vale ressaltar que a utilização da abordagem proposta poderia complementar a utilização de ambas, desta forma, proporcionando um ambiente ainda mais robusto para o desenvolvimento distribuído de aplicações.

6. Conclusão

Este artigo apresentou um sistema de visualização da evolução de software conhecido como EvolTrack. Também sugeriu que através de sua utilização em um ambiente de desenvolvimento distribuído, alguns benefícios devem ser observados: aumento da percepção global do projeto por parte de seus membros; melhor precisão na identificação do grau de evolução do projeto; possibilidade de verificação manual da convergência/divergência do projeto com o que fora especificado; e facilidade na identificação de responsáveis por partes do software. Entretanto, uma das limitações do trabalho é a falta de 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 aumentaria ainda mais a capacidade de percepção da evolução do projeto.

Atualmente, a abordagem está sendo estendida para suportar a integração com o sistema de versionamento Subversion. Já foi criado um Conector de Fonte de Dados Complexo contendo um extrator capaz de resgatar as informações do Subversion e, no presente momento, um transformador capaz de realizar engenharia reversa de código Java para modelos UML está sendo desenvolvido. Note que, devido a flexibilidade da infra-estrutura e da forma como este tipo de conector foi pensado, no futuro poderão ser criados diferentes tipos de transformadores utilizando o mesmo extrator, criado uma única vez. A princípio, cada transformador será responsável por interpretar uma linguagem diferente. Com isso, tudo que o usuário precisaria fazer é verificar através da interface os tipos de transformadores instalados para o Conector de Fonte de Dados de Subversion e adicioná-los ou retirá-los de sua cadeia de transformações, conforme discutido na seção 3. Porém, a ausência desta conexão com o Subversion representa, no presente momento, uma importante limitação da ferramenta, visto que sua utilização em projetos reais se torna bastante restrita sem que exista esta integração. Futuramente, um experimento será preparado com o intuito de prover evidências que suportem as conclusões inferidas pela utilização da abordagem. Um vídeo de demonstração de uso, assim como a ferramenta em si, está disponível para *download* em **http://reuse.cos.ufrj.br/evoltrack/downloads.htm**.

7. Agradecimentos

Os autores gostariam de agradecer à CAPES pelo apoio financeiro na realização deste trabalho.

8. Referências

[1] BALL, A.T, EICK, S.G., "Software Visualization in the Large", IEEE Computer, vol. 29, n. 4, pp. 33-43, Abril, 1996.

[2] BOOCH, G., RUMBAUGH, J., JACOBSON, I., "The Unified Modeling Language User Guide", 2nd ed., Addision-Wesley Proffisional, 2005.

[3] BRUEGGE, B., DUTOIT, A.H., WOLF, T., "Sysiphus: Enabling informal collaboration in global software development", International Conference on Global Software Engineering (ICGSE'06), Florianópolis, Brasil, Outubro, pp. 139-148, 2006.

[4] CEDERQVIST, P. *et al.*, *Version Management with CVS*, 1ª Ed., EUA, Network Theory Ltd, 2006.

[5] CHENG, L., HUPFER, S., ROSS, S., PETTERSON, J., "Jazzing Up Eclipse with Collaborative Tools",

In: Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology Exchange (Anaheim, CA, Oct. 27, 2003), ACM Press, New York, pp. 45-49, 2003.

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

[7] DIEHL, S.,, "Software Visualization – Visualizing the Structure, Behaviour and Evolution of Software", Springer, New York, 2007.

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

[9] FROEHLICH, J., DOURISH, P., "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, 2004.

[10] GERMAN, D.M., HINDLE, A., JORDAN, N., "Visualizing the Evolution of Software using SoftChange", International Journal of Software Engineering and Knowledge (IJSEKE), vol. 16, n. 1, pp. 5-21, Fevereiro, 2006.

[11] HERBSLEB, J.D. & GRINTER, R.E., Splitting the organization and integrating the code: Conway's law revisited. Em Proceedings of the 21st international conference on Software engineering. Los Angeles, California, United States: IEEE Computer Society Press, pp. 85-95, 1999.

[12] HERBSLEB, J.D. & MOCKUS, A., An Empirical Study of Speed and Communication in Globally Distributed Software Development . IEEE Transactions on Software Engineering, 29, 481-494, 2003.

[13] MOCKUS, A. & HERBSLEB, J.D., Challenges of Global Software Development. Em IEEE 7th International Software Metrics Symposium. Londres, Reino Unido, pp. 182-184, 2001.

[14] MURTA, L., CORREA, C., PRUDENCIO, J.G., WERNER, C., "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, 2008.

[15] LEVKOWITZ, H., HERMAN, G.T., "Color Scales for Image Data", IEEE Computer Graphics and Applications, 12(1), pp. 72-80, 1992.

[16] 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.

[17] ROBERTSON, G., CARD, S., MACKINLAY, J., "Information visualization using 3-D interactive animation", Communications of the ACM, vol. 36, n. 4, pp. 57-71, Abril, 1993.

[18] SUSSMAN, B.C., FITZPATRICK, B.W., PILATO, C.M., Version Control with Subversion, 1a Ed., Stanford, CA, EUA, O'Reilly, 2004.