Uma Ferramenta de Apoio ao Controle de Versão das Aplicações Criadas por um Framework

Maria Istela Cagnin*, José Carlos Maldonado, Rosana T. V. Braga, Fernão Germano
Universidade de São Paulo,
Instituto de Ciências Matemáticas e de Computacão,
São Carlos, São Paulo, Brasil,
Caixa Postal 668, CEP 13560-970,
{istela, jcmaldon, rtvb, fernao}@icmc.usp.br

Rosângela Penteado
Universidade Federal de São Carlos,
Departamento de Computacão,
São Carlos, São Paulo, Brasil,
Caixa Postal 676, CEP 13.565-905,
{rosangel}@dc.ufscar.br

Abstract

Framework based application development is increasingly being adopted by software organizations. Frameworks provide reuse of both software design and code, and supply more trustable applications, as the components used to implement them have been previously tested. However, version control is more problematic than in conventional software development, as it is necessary to control both the framework versions and the versions of the applications created with it. Furthermore, aiming to minimize the impact of system requirement changes, framework based software development and reengineering processes adopt the incremental approach, which is a "must" in agile methodologies. This approach makes easier to fulfill requests for system requirements change at any time during the process application. In that context, there is a lack of tools that support version control of applications created with frameworks. This paper presents a tool that aims to aid in the fulfillment of that need, contributing to quality assurance of the products that result from software development or reengineering.

Keywords: Frameworks, Incremental Reengineering, Incremental Development, Version Control Tool.

Resumo

O desenvolvimento de aplicações baseado em frameworks vem sendo praticado cada vez mais pelas empresas de software. Frameworks proporcionam reuso de projeto e de código, e fornecem aplicações mais confiáveis, uma vez que seus componentes, utilizados para construí-las, foram testados anteriormente. No entanto, a problemática do controle de versão é mais grave do que no desenvolvimento de software convencional, pois é necessário controlar tanto as versões do framework quanto as das aplicações por ele criadas. Além disso, visando minimizar o impacto das mudanças que ocorrem nos requisitos do sistema, processos de desenvolvimento e de reengenharia de software baseados em frameworks adotam a abordagem incremental, que é indispensável a metodologias ágeis. Essa abordagem facilita o atendimento das solicitações de mudanças nos requisitos do sistema, em qualquer momento da aplicação do processo. Nesse contexto, há carência de ferramentas que apóiem o controle de versão das aplicações criadas por frameworks. Este artigo apresenta uma ferramenta que tem como objetivo amenizar essa carência colaborando para a garantia da qualidade do produto resultante do desenvolvimento ou da reengenharia de software.

Palavras-Chaves: Frameworks, Reengenharia Incremental, Desenvolvimento Incremental, Ferramenta de Controle de Versão.

*Apoio Financeiro da FAPESP - nr. 00/10881–4

1 Introdução

O software evolui constantemente para se adequar às mudanças administrativas, funcionais, do negócio, governamentais, dentre outras, visando a satisfazer às necessidades dos usuários. Para controlar tais mudanças, é necessário estabelecer um controle das versões produzidas e entregues aos usuários de maneira sistemática e efetiva. Para isso, é necessário considerar durante todo o processo de desenvolvimento/reengenharia de software o *Software Configuration Management - SCM*, que é um conjunto de atividades que gerencia mudanças durante todo o ciclo de vida do software e é um elemento importante para garantir a sua qualidade [23].

O desenvolvimento de aplicações baseado em frameworks [27, 12] vem sendo praticado cada vez mais pelas empresas de software. Um framework é um conjunto de classes que incorpora um projeto abstrato e reusável de soluções para uma família de problemas relacionados em um domínio particular [18], proporcionando reuso de projeto e de código, e fornecendo aplicações mais confiáveis. Isso porque seus componentes, utilizados para construir as aplicações, foram testados anteriormente. Além disso, Fayad e Schmidt afirmam que aplicações podem ser desenvolvidas muito mais rapidamente e com mínimo esforço [12].

No entanto, a problemática de controle de versão no contexto de desenvolvimento baseado em framework é mais grave do que no desenvolvimento de software convencional, pois é necessário controlar as versões do framework e também as aplicações geradas a partir dele.

O controle de versão de frameworks deve ser realizado com muita precaução, pois evoluções em frameworks podem mudar o seu projeto e, conseqüentemente, o das suas aplicações dependentes. Como resultado, essas aplicações podem não aderir ao novo projeto e podem deixar de fornecer o comportamento desejado.

Visando minimizar o impacto das mudanças que ocorrem nos requisitos do sistema, alguns processos de desenvolvimento e de reengenharia de software, baseados em frameworks, adotam a abordagem incremental, que é indispensável a metodologias ágeis [1, 2, 30]. Essa abordagem facilita o atendimento das solicitações de mudanças nos requisitos do sistema, em qualquer momento da aplicação do processo.

Nesse contexto, há carência de ferramentas que apóiem tanto o controle de versão de frameworks quanto o controle de versão de aplicações criadas por eles. A instanciação de frameworks permite que novos componentes sejam adicionados e/ou retirados de uma aplicação criada previamente. No entanto, caso a aplicação tenha sido modificada com a adição manual de novas funcionalidades, não fornecidas pelo framework, e houver necessidade de instanciar o framework posteriormente para adicionar outros componentes na aplicação, todas as linhas de código inseridas manualmente serão perdidas.

Este artigo apresenta uma ferramenta, específica para o controle de versões das aplicações de um determinado framework, que tem como objetivo amenizar essa carência e, também, apoiar na realização da tarefa *controle de versão* da atividade SCM, colaborando para garantir a qualidade do produto resultante tanto no desenvolvimento quanto na reengenharia.

Na Seção 2, discutem-se os trabalhos relacionados. Na Seção 3 apresenta-se a ferramenta, denominada *GREN-WizardVersionControl*, quanto aos aspectos de modelagem conceitual, de arquitetura e de implementação. Na Seção 4 relata-se experiência de uso da ferramenta em um estudo de caso de reengenharia. Na Seção 5, discutem-se as conclusões e os trabalhos futuros.

2 Trabalhos Relacionados

O SCM é considerado como uma das atividades importantes para atingir a certificação de qualidade em diversos padrões, como ISO 9000 [16], *Capability Maturity Model* (CMM) [22], *Capability Maturity Model Integration* (CMMI) [8] e ISO/IEC 15504 (*Software Process Improvement and Capability dEtermination*) ou SPICE [17]. De acordo com Pressman [23], SCM é composto por cinco tarefas: (1) identificação das mudanças, (2) controle de versão, (3) controle de mudanças, (4) auditoria de configuração, e (5) relatórios (que apresentam o que aconteceu, quem fez a mudança, quando aconteceu a mudança, o que será afetado com a mudança, etc). A tarefa *controle de versão* é a mais conhecida e a que possui a principal responsabilidade, pois trata do armazenamento e da recuperação de diferentes versões dos artefatos gerados durante o processo de software.

Algumas características desejáveis de um sistema de SCM são ressaltadas por Midha [20]: a) facilidade de uso, para que o usuário possa utilizar as funcionalidades da ferramenta como parte da execução de seu trabalho; b) facilidade de administração da ferramenta; c) suporte ao desenvolvimento distribuído, para que o sistema possa ser utilizado também por equipes de desenvolvimento distribuídas remotamente; e d) integração da funcionalidade de SCM com outras ferramentas.

A fim de minimizar o espaço de armazenamento das versões dos artefatos no repositório, os sistemas de SCM utilizam **delta** *scripts*, ou somente **delta**, no qual somente uma versão do artefato é armazenada integralmente, e as demais versões possuem apenas as diferenças armazenadas [25]. Existem duas abordagens para o tratamento do delta: **delta negativo** e **delta positivo**. A primeira abordagem, também chamada de **delta reverso**, armazena integralmente a versão mais recente e as diferenças até então, disponibilizando de forma mais rápida a última versão.

A abordagem **delta positivo** armazena integralmente a versão mais antiga e as diferenças desde então. A ferramenta apresentada neste artigo armazena apenas as modificações realizadas em cada versão das aplicações. Isso porque existe um histórico de instanciação do framework, que está sendo considerado pela ferramenta apresentada neste artigo, que armazena as funcionalidades que compõem cada aplicação gerada. Com isso, é possível obter automaticamente informação da versão mais antiga da aplicação.

Existem diversas ferramentas de SCM, com várias funcionalidades, diferentes complexidades e preços. Portanto, é necessário que o engenheiro de software avalie cada uma e escolha a mais apropriada para as suas necessidades. *ClearCase* [15], *Continuus/CM* [28], *Visual Source Safe* [19], *QVCS* [24], *FreeVCS* [14], *CVS* [11], *VersionWeb* e *PVCS* [26] são algumas das ferramentas de SCM existentes. Nenhuma delas atende a necessidade de controlar versões de aplicações geradas por frameworks. No entanto, foi encontrada uma ferramenta, proposta por Tourwé [29], com objetivo diferente da ferramenta apresentada neste artigo, que controla as mudanças causadas na evolução de frameworks e das aplicações criadas a partir deles. Tal ferramenta avalia o impacto que as mudanças podem ter, tanto na hierarquia de classes do framework quanto na das aplicações dependentes dele. Para isso: a) fornece previamente a definição da propagação de mudanças, permitindo que o desenvolvedor avalie o impacto das transformações no framework e de suas aplicações dependentes e detecte possíveis conflitos de "*merge*"; b) sugere como esses problemas podem ser resolvidos.

3 Ferramenta *GREN-WizardVersionControl*

A necessidade de criar a ferramenta *GREN-WizardVersionControl* foi evidenciada em um estudo de caso de reengenharia de um sistema legado de controle de biblioteca [6, 9], e em outro referente ao controle de oficina eletrônica [7], ambos desenvolvidos em Clipper. A reengenharia desses sistemas foi realizada com o apoio do processo de reengenharia ágil PARFAIT [1] [7].

PARFAIT é incremental e iterativo, pois o engenheiro de software tem a flexibilidade de retornar a passos do processo anteriormente executados, a fim de refinar os artefatos produzidos. Além disso, é considerado um processo ágil, pois atende a diversas *práticas* de metodologias ágeis [2]: a) fornece aos usuários uma versão do novo sistema o mais rápido possível – atende à *prática "pequenas releases" do XP (eXtreme Programming)*; b) os usuários participam ativamente da maioria das atividades do projeto de reengenharia e aprovam o produto à medida que o projeto evolui. Em cada interação com os usuários, melhorias no produto são realizadas – atende à *prática "clientes presentes" do XP*; c) testes no novo sistema são realizados constantemente e comparados com os resultados dos testes do sistema legado – atende à *prática "testes constantes" do XP*; d) o planejamento da reengenharia é refeito a cada iteração do processo – atende à *prática "jogo do planejamento" do XP*; e e) incentiva a *prática "programação em pares" do XP*.

PARFAIT utiliza o framework GREN ("Gestão de REcursos de Negócios") [5] como apoio computacional: a) para as atividades de engenharia reversa: na elicitação de novos requisitos e na identificação de regras de negócio do sistema legado; e b) para as atividades de engenharia avante, na criação do protótipo do novo sistema. A construção desse framework foi baseada na linguagem de padrões de análise GRN (Gestão de Recursos de Negócios) [3], portanto facilita a reengenharia de sistemas legados no domínio de "Gestão de Recursos de Negócios". O framework foi construído utilizando a linguagem de programação Smalltalk e o SGBD MySQL [21].

Para facilitar a instanciação do GREN, há um ferramenta de apoio, denominada GREN-Wizard [4], que solicita os padrões da GRN usados na modelagem do sistema e gera as classes da nova aplicação em Smalltalk e as tabelas no banco de dados relacional MySQL.

Após a criação da aplicação de controle de biblioteca, durante a reengenharia com o apoio do GREN-Wizard, algumas alterações foram realizadas em seu código fonte a fim de torná-la funcionalmente mais semelhante ao legado.

Além disso, o cliente solicitou a adição de uma funcionalidade que não estava presente no sistema legado, que foi a de *"Cobrança de Taxa de Multa"* na devolução do livro, caso o usuário devolva o livro com atraso. Tal funcionalidade é suportada por um dos padrões da GRN e, portanto, é fornecida pelo framework GREN. Logo, não precisaria ser implementada manualmente. Com isso, bastaria apenas criar uma nova versão da aplicação no GREN, adicionando-lhe a funcionalidade solicitada. Mas, isso não era possível pois, com uma nova instanciação do framework, todas as linhas de código fonte, inseridas anteriormente na aplicação, seriam perdidas. Dessa forma, evidenciou-se a necessidade de criar uma ferramenta que pudesse armazenar, em uma meta-base de dados, todas as alterações realizadas no código fonte das aplicações geradas pelo framework para que, em uma próxima instanciação do framework, todas as modificações realizadas no código fonte, em versões anteriores, pudessem ser incorporadas na nova versão da aplicação.

Além de evidenciar a necessidade de criar a ferramenta de controle de versão das aplicações criadas pelo framework GREN, observou-se também a necessidade de evoluir a ferramenta de instanciação GREN-Wizard, para que ela pudesse incorporar, durante a criação da nova versão da aplicação, as alterações realizadas no código fonte das versões anteriores. Isso possibilita que o framework GREN apóie efetivamente os processos de desenvolvimento e os

[1] Processo Ágil de Reengenharia baseado em FrAmework no domínio de sistemas de Informação com VV&T

de reengenharia incremental, tornando possível adicionar novas funcionalidades, em qualquer momento da aplicação desses processos.

Assim, a ferramenta foi criada visando que a agilidade do processo de reengenharia PARFAIT não seja comprometida, já que usa a abordagem incremental na fase de implementação. Essa ferramenta é útil não apenas na aplicação do PARFAIT como também no desenvolvimento de novas aplicações de modo incremental.

3.1 Modelagem Conceitual

Esta seção tem como objetivo apresentar as funcionalidades da ferramenta *GREN-WizardVersionControl* e o seu meta-modelo, por meio de um diagrama de casos de uso e de um diagrama de classes, respectivamente. Ambos diagramas são provenientes da UML (*Unified Modelling Language*) [13].

3.1.1 Funcionalidades da Ferramenta

O diagrama de casos de uso da ferramenta *GREN-WizardVersionControl* é mostrado na Figura 1. Todos os casos de uso, com exceção do caso de uso com preenchimento cinza ("Decidir conflito entre métodos"), são executados durante as modificações no código fonte da aplicação gerada pelo framework. O caso de uso "Decidir conflito entre métodos" é executado durante a instanciação do framework com o apoio da ferramenta GREN-Wizard, quando algum método herdado do framework foi modificado pelo engenheiro de aplicação [2]. O caso de uso "Alterar atributo" é somente executado para atributos primitivos (`inteiro, string, float, date`) e não herdados do framework. Além de adicionar atributos primitivos (caso de uso "Adicionar Atributos") é possível adicionar atributos do tipo enumerado (caso de uso "Adicionar atributo enumerado") e multivalorado (caso de uso "Adicionar atributo multivalorado"), que possuem tratamentos especiais. Os casos de uso "Remover classe", "Remover atributo", "Remover categoria método" e "Remover método" são executados somente para classes, atributos, categoria de métodos e métodos não herdados do framework, respectivamente.

As restrições de remoção foram estabelecidas a fim de garantir que os componentes herdados do framework (ou seja, classes, atributos, métodos, etc) não sejam removidos acidentalmente ou propositadamente do código fonte da aplicação criada pelo framework. Só é permitido que o engenheiro de aplicação sobreponha ou modifique os métodos herdados, garantindo assim que nenhum *hot-spot* [3] seja danificado, garantindo que a estrutura da aplicação esteja correta e funcionando adequadamente.

3.1.2 Meta-Modelo da Ferramenta

O meta-modelo da ferramenta *GREN-WizardVersionControl* é apresentado por meio de um diagrama de classes, como ilustrado na Figura 2.

Algumas classes existentes do framework GREN (`PersistentObject`) e da ferramenta de instanciação GREN-Wizard (`GrenWizardCodeGenerator`, `GrenWizardGUI`, `GrenApplication`, `AttributeMappingForm` e `AttributeListForm`) tiveram que ser modificadas, conforme ilustrado no diagrama as classes com preenchimento. As classes `ClassVersionControl`, `ClassProtocolVersionControl` e `ClassMethodVersion-Control` contém informações relevantes sobre a classe que está sendo atualizada, sobre o protocolo ou categoria [4] do método e sobre o método, respectivamente. Como essas classes contém algumas informações em comum, elas herdam da classe `MetaVersionControl`. A classe `GREN-WizardVersionControl` faz todo o controle de versão e contém uma interface semelhante ao *System Browser*, que é uma das ferramentas do VisualWorks [10] utilizada para realizar a programação em Smalltalk. A classe `GrenWizardDifferatorTool` [5] mostra os trechos de código diferentes dos métodos herdados do framework mas que foram modificados. Isso é feito durante a instanciação de uma nova versão da aplicação por meio da ferramenta GREN-Wizard.

3.2 Arquitetura

A arquitetura da ferramenta *GREN-WizardVersionControl* foi baseada na arquitetura do framework GREN e da ferramenta de instanciação GREN-Wizard.

A arquitetura do GREN foi projetada em três camadas: a de persistência, a de negócios e a de interface gráfica com o usuário (GUI), conforme apresentado na Figura 3. A **camada de persistência** possui classes que tratam da conexão com a base de dados, gerenciamento dos identificadores dos objetos e persistência dos objetos. A **camada**

[2] suas funções estão apresentadas na Seção 3.2

[3] estruturas variáveis do framework que contém os componentes que podem ser adaptados e estendidos pelo engenheiro de software na aplicação instanciada a partir do framework, para se adequarem às necessidades de um determinado sistema (por exemplo, regras do negócio, requisitos funcionais inerentes às políticas internas da empresa, etc).

[4] em Smalltalk, métodos da classe são organizados em protocolos (ou categorias) que são agrupamentos de métodos semanticamente relacionados.

[5] baseada na classe `Differator` do VisualWorks

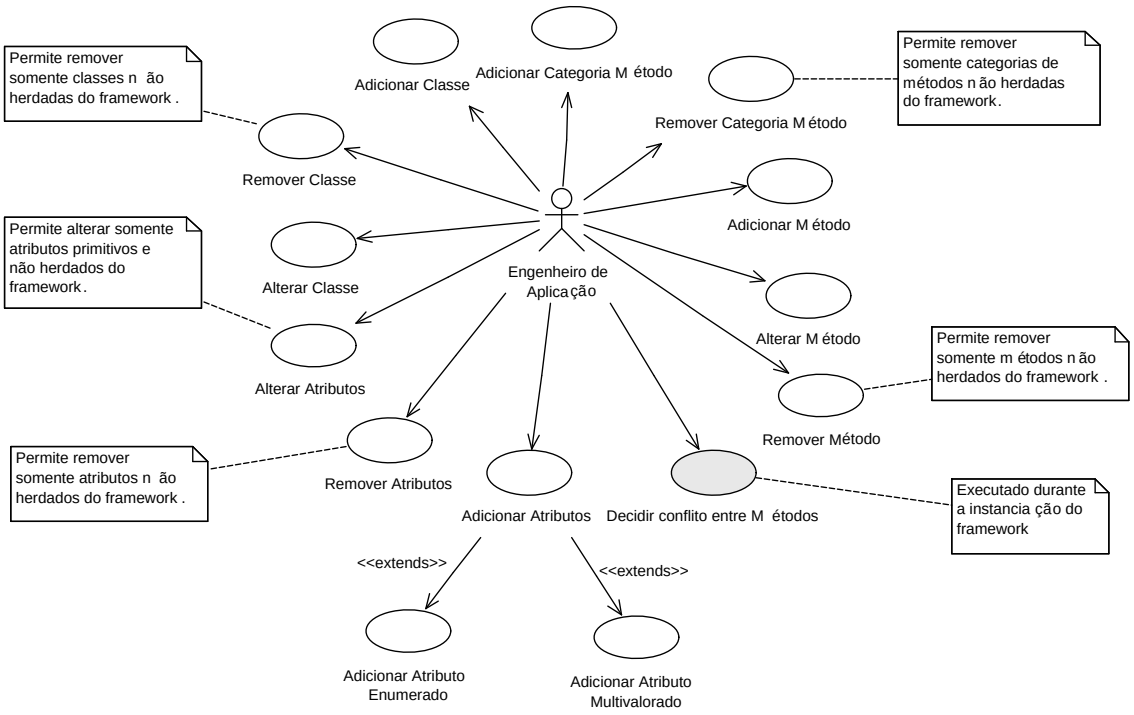

Figura 1: Diagrama de Casos de Uso da ferramenta *GREN-WizardVersionControl*

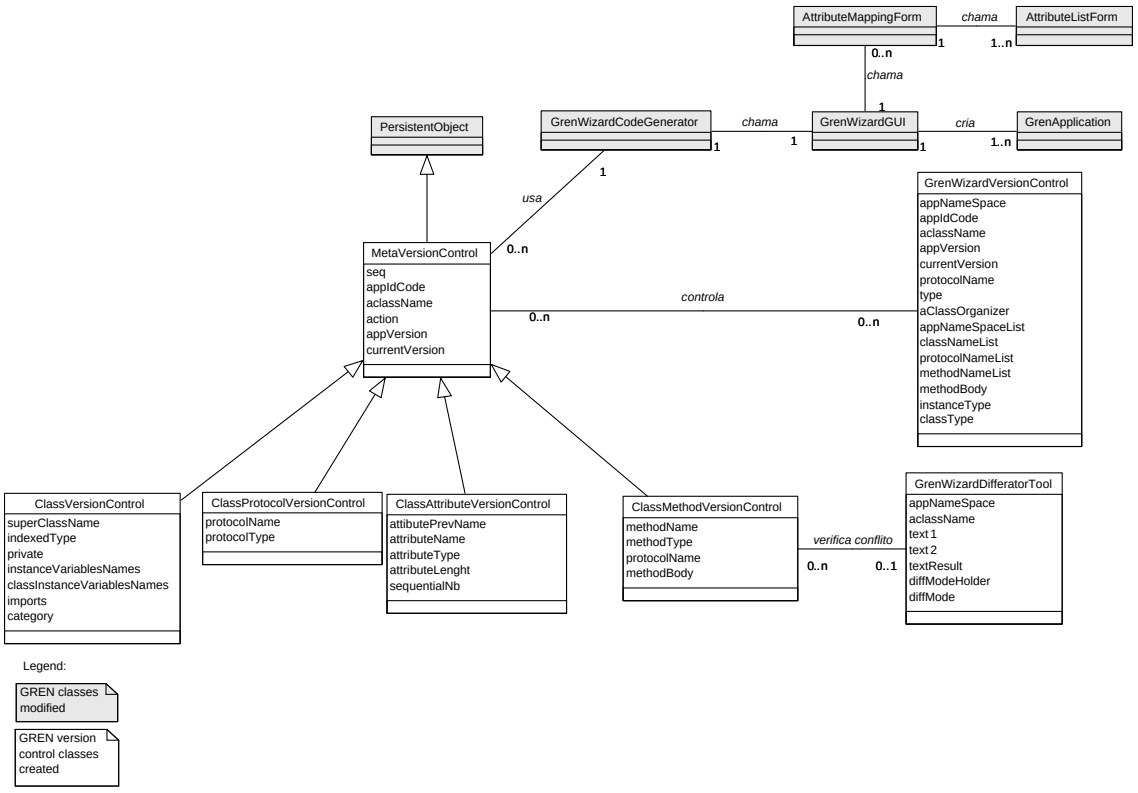

Figura 2: Diagrama de Classes da ferramenta *GREN-WizardVersionControl*

de negócios comunica-se com a camada de persistência para armazenar um objeto. Na camada de negócios existem diversas classes derivadas diretamente dos padrões de análise que compõem a linguagem de padrões GRN, ou seja, as classes e relacionamentos contidos em cada padrão possuem a implementação correspondente nesta camada. A **camada de interface gráfica com o usuário** contém as classes que tratam a entrada e a saída de dados, permitindo a interação do usuário final com o sistema. Essa camada comunica-se com a camada de negócios para obtenção de objetos a serem mostrados na interface com o usuário e para devolver informações a serem processadas pelos métodos da camada de negócios.

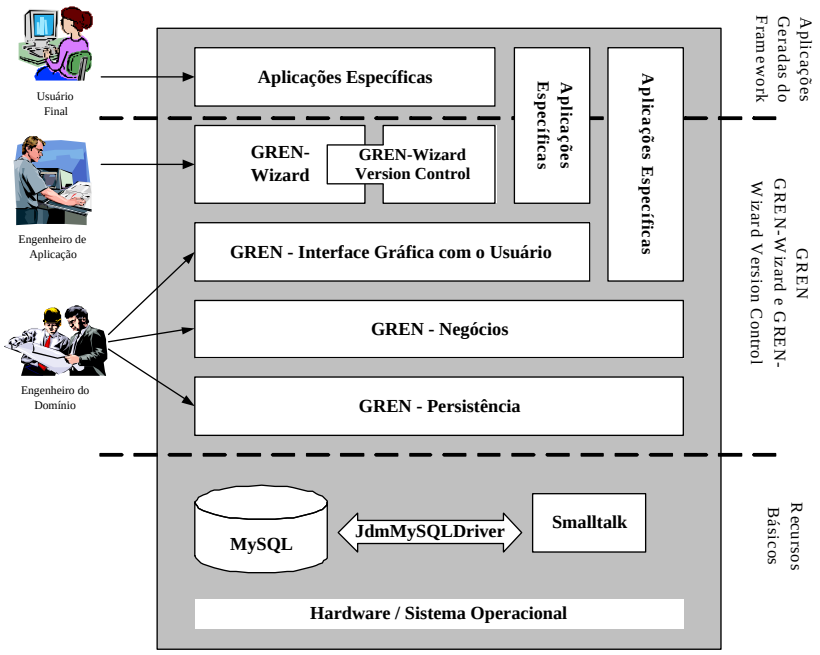

Figura 3: Arquitetura do framework GREN (adaptada de [5])

A camada GREN-Wizard, que gera o código da aplicação a partir da sua especificação baseada nos padrões da GRN, situa-se acima da camada do GREN, pois utiliza todas as demais camadas. A camada GREN-Wizard comunica-se com a camada *GREN-WizardVersionControl*, que é responsável pelo controle das versões das aplicações instanciadas e apóia o desenvolvimento e reengenharia incremental e, conseqüentemente, processos baseados em metodologias ágeis. Aplicações específicas podem ser construídas a partir da camada de interface gráfica com o usuário, sem o apoio da camada GREN-Wizard, usando (por meio de herança ou referência a objetos) classes de todas as camadas do GREN inferiores a essa, ou podem ser construídas com base na camada de negócios, caso a camada de interface gráfica com o usuário não seja reutilizada a partir do GREN, mas implementada separadamente.

A Figura 4 mostra a arquitetura das ferramentas GREN-Wizard e *GREN-WizardVersionControl*. Dois atores interagem diretamente com a ferramenta GREN-Wizard: o engenheiro do domínio, responsável pela construção do GREN-Wizard (e, provavelmente, também pelo desenvolvimento da linguagem de padrões e por parte da construção do framework) e o engenheiro de aplicações, que utiliza o GREN-Wizard para gerar aplicações específicas por meio de uma interface gráfica com o usuário (GUI). Essa interface facilita a especificação da aplicação de acordo com os padrões da linguagem de padrões GRN. O usuário final interage indiretamente com a ferramenta GREN-Wizard, pois executa o código da aplicação específica, gerado por ela.

Três módulos compõem a arquitetura do GREN-Wizard: o módulo de especificação do domínio, o módulo de especificação da aplicação e o módulo de geração de código.

O módulo de **especificação do domínio** permite a representação de informações sobre a linguagem de padrões GRN e seu mapeamento para as classes do framework GREN. O módulo de **especificação da aplicação** é responsável por armazenar as informações sobre a modelagem das aplicações específicas usando a linguagem de padrões. Por fim, o módulo de **geração de código** baseia-se nas informações disponibilizadas sobre a especificação do domínio e sobre a especificação da aplicação, para gerar o código específico da aplicação, que deve juntar-se ao código do framework para produzir a aplicação a ser entregue ao usuário final.

De acordo com a Figura 4, o engenheiro de aplicação deve utilizar inicialmente a ferramenta GREN-Wizard para gerar uma primeira versão da aplicação. Nessa ferramenta, como mencionado anteriormente, ele especifica a aplicação informando os padrões da GRN utilizados para modelá-la. Qualquer modificação manual no código fonte da aplicação, a fim de inserir novas funcionalidades não fornecidas pelo framework, deve ser realizada por meio da ferramenta *GREN-WizardVersionControl*, que armazena cada modificação em uma meta-base de dados. Quando houver

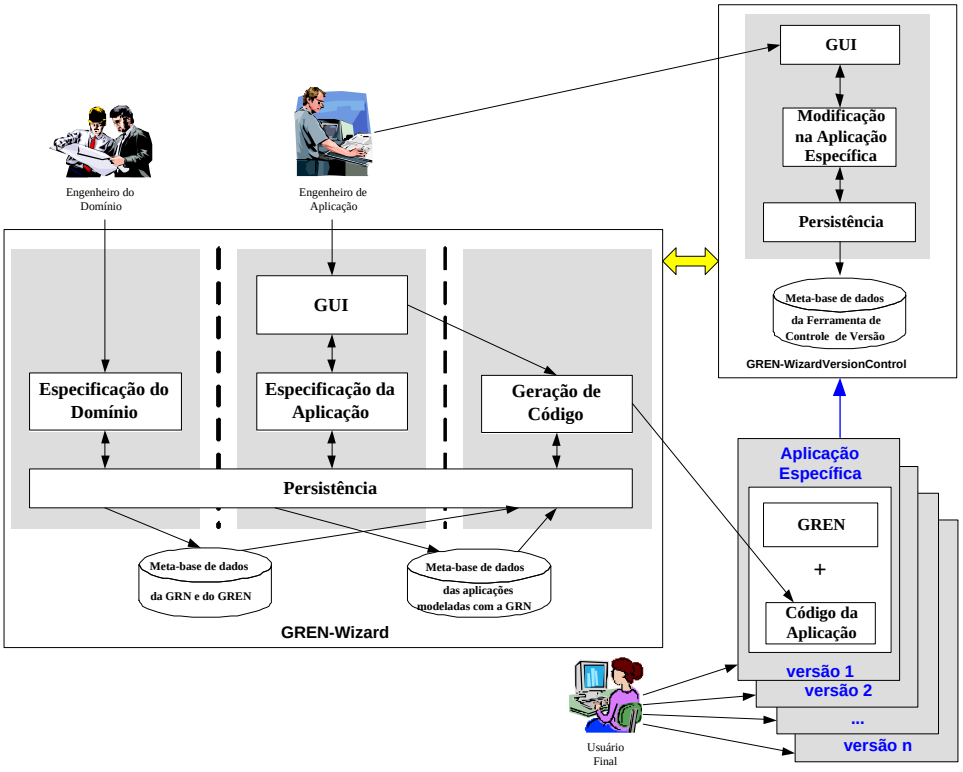

Figura 4: Arquitetura das ferramentas GREN-Wizard e GREN-WizardVersionControl

necessidade de inserir novas funcionalidades na aplicação, fornecidas pelo framework, o engenheiro de aplicação deve utilizar a ferramenta GREN-Wizard. Essa ferramenta gera a nova versão da aplicação, com as novas funcionalidades herdadas, juntamente com todas as modificações realizadas anteriormente no código fonte.

O usuário final interage com todas as versões da aplicação liberadas para uso, ou seja, as *releases*.

3.3 Implementação

A ferramenta *GREN-WizardVersionControl* foi implementada na mesma linguagem de programação em que o framework GREN e a ferramenta GREN-Wizard estão implementados, ou seja, Smalltalk (ambiente VisualWorks), para que essas ferramentas pudessem ser integradas da melhor maneira possível. Isso atende uma das características desejáveis de um sistema de SCM ressaltadas por Midha [20]. O meta-modelo foi mapeado para uma meta-base de dados no SGBD MySQL [21].

A Figura 5 apresenta a tela principal da ferramenta *GREN-WizardVersionControl*, cujo projeto de interface foi baseado na tela do *System Browser* do VisualWorks, a fim de torná-la o mais similar possível, objetivando facilitar o uso da ferramenta e possibilitar que o usuário utilize as funcionalidades da ferramenta como parte da execução de seu trabalho. Isso atende outra característica desejável de um sistema de SCM [20].

A administração da ferramenta *GREN-WizardVersionControl* é fácil, colaborando para que mais uma característica desejável de um sistema de SCM [20] seja alcançada.

Como mencionado anteriormente, além da criação da ferramenta *GREN-WizardVersionControl* foi necessário também alterar a ferramenta GREN-Wizard, para que as alterações realizadas pelo engenheiro de software no código fonte de uma determinada aplicação fossem consideradas também nas próximas instanciações do framework para tal aplicação. Na geração de uma nova versão da aplicação pelo GREN-Wizard, as classes, métodos e atributos que foram removidos, inseridos ou atualizados na versão anterior da aplicação com o apoio da ferramenta *GREN-WizardVersionControl* e que estão armazenados em sua meta-base de dados, são detectados. As inserções e remoções de código fonte são resolvidas automaticamente pela ferramenta GREN-Wizard. No entanto, as modificações nos métodos herdados do framework são identificadas pela ferramenta como conflito entre o código fonte do método do framework e o da aplicação e deve ser resolvido pelo engenheiro de software em tempo de execução do GREN-Wizard. Para isso, quando a ferramenta GREN-Wizard detecta conflitos, é apresentada uma tela para o engenheiro da aplicação, Figura 6 (lado esquerdo), que deve informar o conteúdo do método que a ferramenta de instanciação deve considerar na versão da aplicação sendo criada. Para isso, é necessário copiar e colar o conteúdo correto na caixa de texto *Resulting Method*, Figura 6 (lado direito). Em seguida, deve-se clicar no botão *Accept* para compilar as linhas de código fonte informadas e para dar prosseguimento a instanciação do framework.

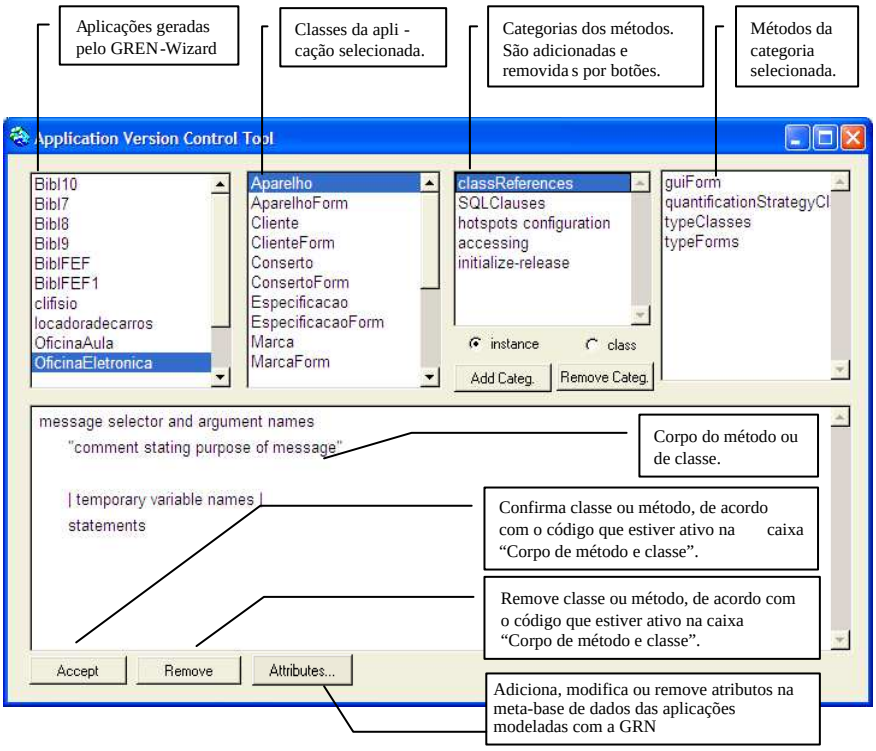

Figura 5: Tela principal da ferramenta *GREN-WizardVersionControl*

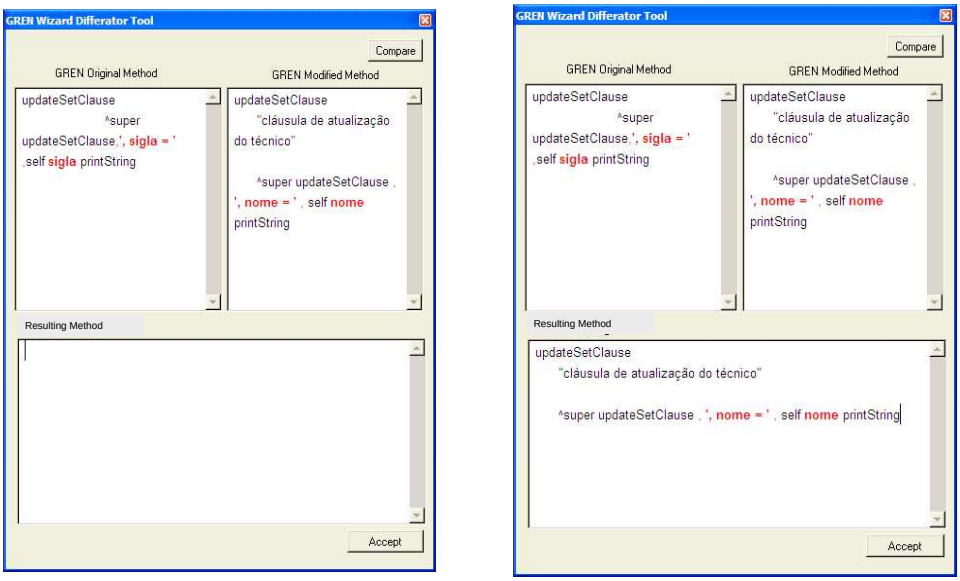

Figura 6: Tela para solução de conflito entre métodos do framework e da aplicação

A ferramenta GREN-Wizard também cria a base de dados e os *scripts* das tabelas em MySQL da aplicação que foi gerada. Uma alteração nessa funcionalidade também foi realizada, permitindo que o engenheiro de aplicação tenha a opção de gerar uma nova base de dados da aplicação ou manter a atual e apenas atualizar a estrutura modificada, a fim de não perder os dados da aplicação inseridos na versão anterior.

A ferramenta *GREN-WizardVersionControl* foi testada em outro estudo de caso de reengenharia do sistema de controle de biblioteca, com o apoio do PARFAIT, e mostrou-se efetiva, como apresentado na próxima seção.

4 Exemplo de Uso da Ferramenta *GREN-WizardVersionControl*

A Figura 7 apresenta a tela "Empréstimo" de livro e o script SQL da tabela `Emprestimo` da primeira versão do sistema de controle de biblioteca, ambos gerados pelo GREN-Wizard durante a reengenharia com o apoio do processo

ágil PARFAIT.

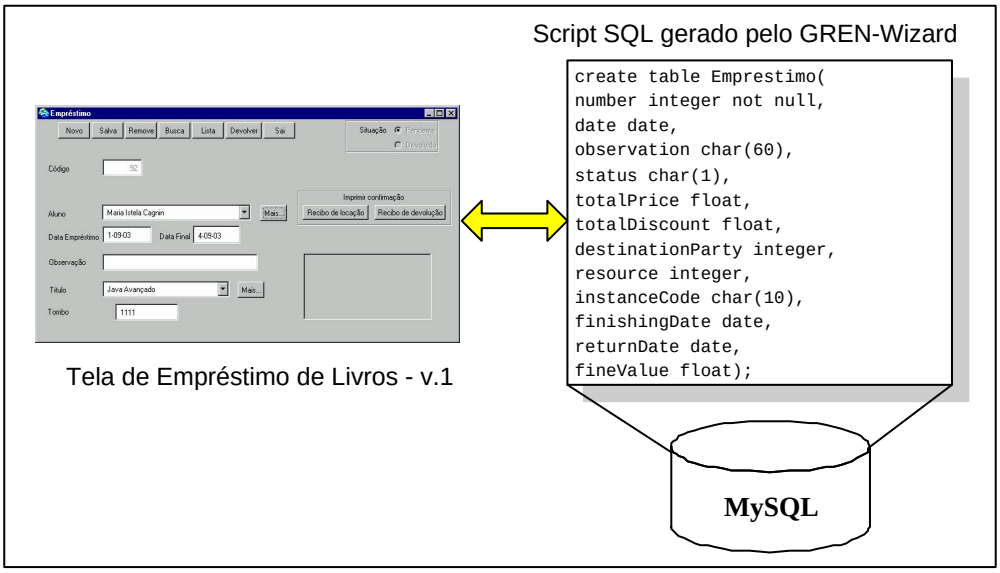

Figura 7: Fragmento da Versão 1 da aplicação de controle de biblioteca

No decorrer da aplicação do processo de reengenharia, observou-se a necessidade de realizar algumas mudanças no código fonte da primeira versão da aplicação. Para isso, utilizou-se a ferramenta *GREN-WizardVersionControl*. A Figura 8 apresenta uma das mudanças realizadas, a qual modificou o conteúdo do método `windowLabel`, herdado do framework GREN: do conteúdo original `^'Emprestimo'` para o conteúdo modificado `^'Empréstimo de Livro'`. Essa modificação foi armazenada na meta-base de dados da ferramenta *GREN-WizardVersionControl*, Figura 8, para que pudesse ser incorporada pelo GREN-Wizard na criação da próxima versão da aplicação.

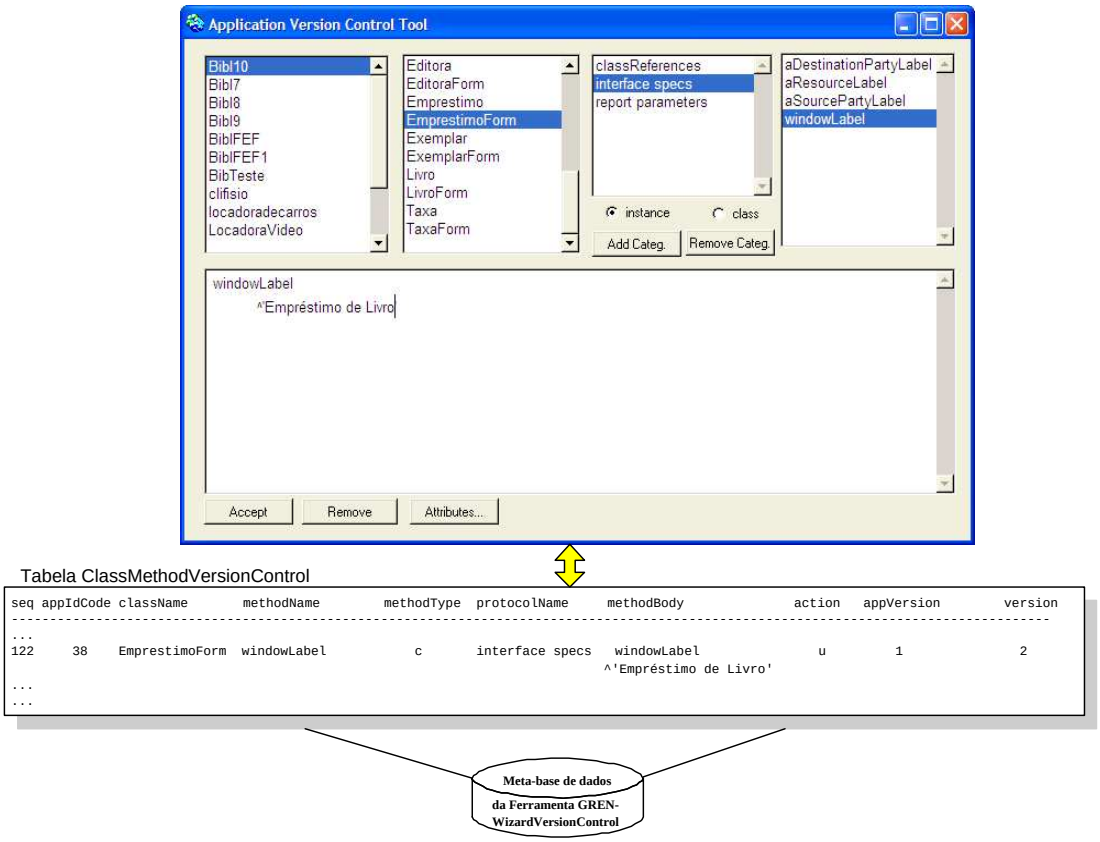

Figura 8: Mudança no código fonte da versão 1 da aplicação de controle de biblioteca

Outra mudança desejada na aplicação foi a adição da funcionalidade *"Cobrança de Taxa de Multa"*. Nesse

caso, utilizou-se a ferramenta GREN-Wizard, uma vez que essa funcionalidade é fornecida pelo GREN. Durante a instanciação da nova versão da aplicação, a ferramenta GREN-Wizard detectou a modificação realizada no método `windowLabel` como conflito entre o conteúdo do método herdado do framework e o da aplicação (Figura 9). Com isso, o engenheiro de aplicação teve que solucionar esse conflito, por meio da tela *GREN-Wizard Differator Tool* (Figura 9), apresentada pelo GREN-Wizard em tempo de execução da instanciação.

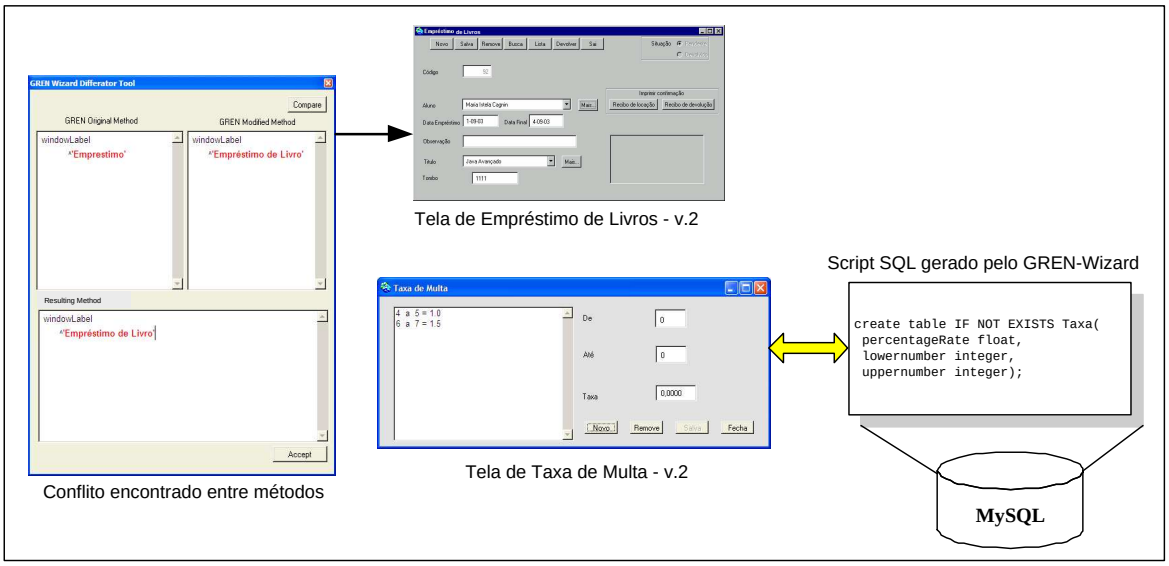

Figura 9: Fragmento da Versão 2 da aplicação de controle de biblioteca

A Figura 9 também apresenta algumas telas da nova versão do sistema de controle de biblioteca: a tela “Empréstimo de Livros”, tendo o título da janela atualizado, e a tela “Taxa de Multa”, resultante da funcionalidade adicionada por meio do GREN-Wizard, bem como o seu respectivo script SQL.

5 Conclusões

Observou-se que a ferramenta *GREN-WizardVersionControl* apóia processos de reengenharia e de desenvolvimento de sistemas baseados em framework, que utilizam abordagem incremental, pois, por meio dela, o engenheiro de aplicação obtém um registro histórico de todas as mudanças feitas em cada nova versão do sistema. Essas mudanças são automaticamente incorporadas às novas versões geradas pelo GREN-Wizard, com o mínimo de intervenção do engenheiro de aplicação.

Outros estudos de caso devem ser realizados com a ferramenta para aprimorar os testes já realizados e adicionar melhorias, se for o caso. Além disso, é necessário utilizá-la em outros contextos, por exemplo, manutenção de software.

Apesar da ferramenta apresentada neste artigo ser específica a um determinado framework, infere-se a possibilidade de utilizar a sua modelagem conceitual e arquitetura para implementar outras ferramentas, com o mesmo propósito, em outros frameworks existentes.

Notou-se que a ferramenta *GREN-WizardVersionControl* é classificada como um sistema de SCM, pois atende a maioria das características desejáveis [20] para esse tipo de sistema: é fácil de utilizar, permite que o usuário utilize as funcionalidades da ferramenta como parte da execução de seu trabalho; é fácil de ser administrada; e é integrada com a ferramenta GREN-Wizard.

Referências

[1] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta. Agile software development methods. review and analysis. ESPOO (Technical Research Centre of Finland)’ 2002. VTT Publications n. 478, 2002. `http://www.inf.vtt.fi/pdf/publications/2002/P478.pdf`. Acessado em: Dezembro,2003.

[2] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2nd edition, 2000.

[3] R. T. V. Braga, F. S. R. Germano, and P. C. Masiero. A pattern language for business resource management. In *PLOP’1999, Conference on Pattern Languages of Programs*, pages 1–33, August 1999.

[4] R. T. V. Braga and P. C. Masiero. Building a Wizard for Framework Instantiation Based on a Pattern Language. In *OOIS'2003, 9th International Conference on Object-Oriented Information Systems*, pages 95–106, Geneva, Switzerland, September 2003. Lecture Notes on Computer Science, LNCS 2817.

[5] R.T.V. Braga. *Um Processo para Construção e Instanciação de Frameworks baseados em uma Linguagem de Padrões de Domínio Específico*. PhD thesis, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2003.

[6] M. I. Cagnin, J. C. Maldonado, F. S. Germano, A. Chan, and R. D. Penteado. Um estudo de caso de reengenharia utilizando o processo PARFAIT. In *SDMS'2003, Simpósio de Desenvolvimento e Manutenção de Software da Marinha*, Niterói, RJ, 2003. CD-ROM.

[7] M. I. Cagnin, J. C. Maldonado, F. S. Germano, and R. D. Penteado. PARFAIT: Towards a framework-based agile reengineering process. In *ADC'2003, Agile Development Conference*, pages 22–31. IEEE, June 2003.

[8] Carnegie Mellon University. Capability maturity model integration - version 1.1. Technical Report CMU/SEI-2002-TR-003, Carnegie Mellon University, Software Engineering Institute, 2002.

[9] A. Chan, M. I. Cagnin, and J. C. Maldonado. Aplicação do processo de reengenharia PARFAIT em um sistema legado de controle de biblioteca. Documento de Trabalho, ICMC-USP, May 2003.

[10] Cincom. Visualworks 5i.4 non-commercial, 2003. `http://www.cincom.com/`. Acessado em: Fevereiro, 2003.

[11] CVS. Concurrent Versions System - The open standard for version control. `http://www.cvshome.org`. Acessado em: Dezembro, 2003.

[12] M. Fayad and D. C. Schmidt. Object-oriented application frameworks. *Communications of the ACM*, 40(10), 1997.

[13] M. Fowler and K. Scott. *UML Distilled - Applying the Standard Object Modeling Language*. Addison-Wesley, first edition, 1997.

[14] FreeVCS. Free Version Control System. `http://www.thensle.de`. Acessado em: Abril, 2004.

[15] IBM. ClearCase. `http://www-306.ibm.com/software/awdtools/clearcase`. Acessado em: Abril, 2004.

[16] ISO 9000. International Organization for Standardization. `http://www.iso.ch/iso/en/iso9000-14000/iso9000/iso9000index.html`. Acessado em: Abril, 2004.

[17] ISO/IEC 15504. International Standard for Software Process Assessment. `http://isospice.com/standard/tr15504.htm`. Acessado em: Abril, 2004.

[18] Ralph Johnson and Brian Foote. Designing reusable classes. *Journal of Object-Oriented Programming*, 1988.

[19] Microsoft. Visual SourceSafe. `http://www.msdn.microsoft.com/vstudio/previous/ssafe`. Acessado em: Abril, 2004.

[20] A. K. Midha. Software configuration management for the 21st century. *Bell Labs Technical Journal*, 2(1), 1997. `http://citeseer.nj.nec.com/midha97software.html`. Acessado em: Fevereiro, 2004.

[21] MySQL. Mysql reference manual, 2003. `http://www.mysql.com/doc/en/index.html`. Acessado em: Dezembro, 2003.

[22] M. C. Paulk. Capability maturity model for software - version 1.1. Technical Report CMU/SEI-1993-TR-24, Carnegie Mellon University, Software Engineering Institute, 1993.

[23] R. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 5th edition, 2001.

[24] Quma Software, Inc. QVCS – Quma Version Control System. `http://www.qumasoft.com`. Acessado em: Abril, 2004.

[25] C. Reichenberger. Delta storage for arbitrary non-text files. *ACM*, pages 144–152, 1991.

[26] Synergex. PVCS. `http://www.pvcs.synergex.com`. Acessado em: Abril, 2004.

[27] Taligent. Building object-oriented frameworks, 1997. http://www.ibm.com/java/education/oobuilding/index.html. Acessado em: Fevereiro, 2002.

[28] Telelogic. Continuus/CM. http://www.telelogic.com/continuus.cfm. Acessado em: Abril, 2004.

[29] T. Tourwé. *Automated Support For Framework-Based Software Evolution*. PhD thesis, Department of Computer Science, Vrije Universiteit Brussel, Brussel, 2002.

[30] D. Turk, R. France, and B. Rumpe. Limitations of agile software processes. In *Third International Conference on Extreme Programming and Agile Processes in Software Engineering (XP'2002)*, pages 43–46, Alghero, Sardinia, Italy, May 2002.