FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO "EURÍPIDES DE MARÍLIA" – UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MAURÍCIO MASSARU ARIMOTO

TOFRA – UMA FERRAMENTA DE CONTROLE DE VERSÕES DE APLICAÇÕES BASEADAS EM *FRAMEWORKS*

MARÍLIA
2006

MAURÍCIO MASSARU ARIMOTO

TOFRA – UMA FERRAMENTA DE CONTROLE DE VERSÕES DE APLICAÇÕES BASEADAS EM *FRAMEWORKS*

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, como requisito parcial para a obtenção do Título de Bacharel em Ciência da Computação.

Orientadora:
Profª. Drª. Maria Istela Cagnin Machado
Co-orientador:
Prof. Dr. Valter Vieira de Camargo

MARÍLIA
2006

MAURÍCIO MASSARU ARIMOTO

TOFRA – UMA FERRAMENTA DE CONTROLE DE VERSÕES DE APLICAÇÕES BASEADAS EM *FRAMEWORKS*

Banca examinadora do Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da UNIVEM/F.E.E.S.R., como requisito parcial para a obtenção do Título de Bacharel em Ciência da Computação.

Resultado: 9,5 (Nove e Meio)

ORIENTADOR: _____
Prof. Dr. Valter Vieira de Camargo

1º EXAMINADOR: _____
Prof. Dr. Márcio Eduardo Delamaro

2º EXAMINADOR: _____
Prof. Drª. Fátima L. S. Nunes Marques

Marília, 04 de Dezembro de 2006.

Aos meus pais, Tsutomu e Suzana
e à minha noiva, Andréia.

AGRADECIMENTOS

Primeiramente a Deus, pela preciosa vida, e pela força que me foi dada para superar todos os obstáculos e alcançar com êxito os meus objetivos.

Aos meus pais, Tsutomu e Suzana, pelo amor, carinho, esforço e dedicação ao longo dos anos, e que apesar de todas as dificuldades, não pouparam esforços para a minha formação. Sem essas pessoas maravilhosas nada seria possível. Meu eterno obrigado.

À minha noiva, Andréia pelo amor, carinho e compreensão nos momentos mais difíceis e pela paciência quando tive que me ausentar para a realização deste importante empreendimento.

À Neusa e ao Benjamim, pelo amor, carinho e respeito e por terem me acolhido nos momentos em que mais precisei. Sou eternamente grato por tudo que fizeram por mim. Meu muito obrigado.

À profª Drª Maria Istela Cagnin Machado que me orientou na primeira fase deste trabalho, pelo apoio, paciência, dedicação, respeito e pelos ensinamentos passados durante esses anos.

Ao prof. Dr. Valter Vieira de Camargo que deu continuidade a este projeto, com muito profissionalismo, dedicação e sempre com boas idéias a acrescentar. Entendeu as minhas limitações e soube conduzir com êxito este projeto.

Aos meus irmãos por me incentivarem a retornar aos estudos e poder cumprir mais uma etapa em minha vida.

Aos professores Auri Vicenzi e Edward David Moreno pelos ensinamentos passados ao longo dos anos e pelo extremo profissionalismo. Foi uma enorme satisfação tê-los como professor.

À todos os professores do Curso de Bacharelado em Ciência da Computação do UNIVEM, em especial aos professores André L. S. Kawamoto, Ildeberto A.Rodello (Beto), Márcio Eduardo Delamaro, Edmundo Sergio Spotto (Dino), Elton Aquinori Yokomizo, Fátima L. S. Nunes Marques, Luiz Fernandes Galante, Maria Cristina, Ana Paula Piovesan M. Peruzza, Antonio Carlos Sementille, Marcos Luiz Mucheroni e Remo Braga, que foram importantes no processo de aprendizado e na minha formação acadêmica.

Aos meus colegas da graduação, Kleber Sampaio, Rafael Henrique, Jeferson Parcket, José Ricardo, Renato Giraldi, Ronaldo, Marcelo Hugo, Claudinei, Celso, Carlos Henrique, Carlos Fuji, Thiago Ishio, Elter, Julio César, Danilo, João Paulo, Rodrigo Coltri, Fabio Massalino, por compartilharem comigo momentos difíceis e de descontração. Valeu galera. Peço desculpas se esqueci de alguém.

À todos os funcionários da Biblioteca, pelo ótimo atendimento prestado.

Por fim, a todos que, de forma direta ou indireta contribuíram para a realização deste trabalho.

You cannot teach a man anything;
You can only help him to find it within himself.
(Galileo Galilei)

Be happy.
It's one way of being wise.
(Colette)

ARIMOTO, Maurício Massaru. **TOFRA – Uma Ferramenta de Controle de Versões de Aplicações baseadas em *Frameworks*.** 2006. 123 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

RESUMO

Um dos desafios da Engenharia de Software é incorporar técnicas para aprimorar o processo de desenvolvimento de software, e conseqüentemente, aumentar a produtividade e a qualidade do software desenvolvido. Nesse contexto, técnicas de reutilização de software como os *frameworks* têm tido um avanço significativo, uma vez que conseguem reunir o conhecimento de um domínio específico, propiciando um maior nível de reúso de software que outras tecnologias. Porém, existem algumas desvantagens, como um controle de versões mais complexo. Isto acontece porque devem ser controladas tanto as versões do *framework*, como as versões das aplicações. Diversas ferramentas de apoio ao controle de versões foram encontradas na literatura. No entanto, nenhuma delas foi desenvolvida especificamente para apoiar o controle de versões de *frameworks* e de aplicações baseadas em *frameworks.* Este trabalho está inserido dentro desse contexto e propõe a implementação de uma ferramenta que apóie o controle de versões de *frameworks*. Uma característica importante é que a ferramenta desenvolvida também permite controlar as versões de *frameworks* orientados a aspectos. Este novo cenário é ainda mais complexo, pois mais de um *framework* pode ser vinculado a uma determinada aplicação, diferentemente dos *frameworks* convencionais. Para auxiliar na compreensão e no desenvolvimento do trabalho foi necessário um estudo acerca das principais características relacionadas aos *frameworks* e gerência de configuração de software. Ressalta-se que para avaliar a ferramenta desenvolvida, estudos de caso foram realizados. Com os experimentos foi possível afirmar que a ferramenta fornece importante contribuição para a gerência de configuração de aplicações baseadas em *frameworks.*

Palavras-chave: *Frameworks*. Gerência de configuração de software. Controle de versões.

ARIMOTO, Maurício Massaru. **TOFRA – Uma Ferramenta de Controle de Versões de Aplicações baseadas em *Frameworks*.** 2006. 123 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

ABSTRACT

One of the challenges of the software engineering is to incorporate techniques into software development process to improve the productivity and increase the quality of the software. In this context, reuse techniques such as frameworks, has been an important contribution since they incorporate the knowledge of a specific domain promoting a better level of software reuse than other technologies. However, the existing advantages bring together some disadvantages, such as a more complicated version control. This occurs because both the versions of frameworks and versions of applications must be controlled. Many tools were found in the literature that supports version control of standard applications, however no one of them had been developed to support version control of frameworks and framework-based applications. This research work fit into this context proposing a tool for supporting the version control of frameworks and framework-based applications. An important characteristic of the tool is that it also allows the version control of applications developed with support of aspect-oriented frameworks. This new scenario is more complicated because more than one framework can be connected to an application, unlike the standard frameworks. A case study involving the main characteristics of frameworks and configuration management was conducted to help in the comprehension and development of the tool. It is worth mentioning that case studies were conducted to validate the tool. It has been possible to say that, the tool provides important contribution for configuration management of framework-based applications.

Keywords: Frameworks. Software Configuration Management. Version Control.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CVS: *Concurrent Version System*

CMM: *Capability Maturity Model*

FOA: *Frameworks* Orientados a Aspectos

FOO: *Frameworks* Orientados a Objetos

FT: *Frameworks* Transversais

JSP: *JavaServer Pages*

OA: Orientada a Aspectos

OO: Orientada a Objetos

OOT: *Object Oriented Technology*

POA: Programação Orientada a Aspectos

PREF: Processo de Evolução de *Frameworks* de Aplicação

RCS: *Revision Control System*

SADT: *Structured Analysis and Design Technique*

SEI: *Software Engineering Institute*

SCCS: *Source Code Control System*

SCM: *Software Configuration Management*

SGBD: Sistema de Gerenciamento de Banco de Dados

SUMÁRIO

INTRODUÇÃO

Contexto

A Engenharia de Software tem adotado diversos modelos para a melhoria do processo de desenvolvimento de software buscando minimizar os custos, diminuir os esforços despendidos, aumentar a produtividade e principalmente garantir a qualidade do produto. Um dos avanços na área é a técnica de reúso de software. Neste contexto, o uso de soluções "semi-prontas", como os *frameworks*, tornam-se fundamentais.

Com a popularização do paradigma orientado a objetos e o aumento de sistemas complexos, a reutilização de software ganhou força. Paralelamente, surgiram os *frameworks*, que possibilitam o reúso de grandes componentes ou estruturas de software em domínios específicos, minimizando o tempo de desenvolvimento (JOHNSON e FOOTE, 1988; BOSCH *et al.,* 1997). Os *frameworks* são tecnologias promissoras para as empresas de software. O reúso de software é um dos fatores que contribuem para o sucesso dessa tecnologia, permitindo que aplicações sejam facilmente geradas a partir da utilização de componentes reutilizáveis implementados nos *frameworks*. Os *frameworks* geralmente seguem padrões bem definidos. Esses padrões permitem que sejam reutilizadas soluções para problemas que muitos desenvolvedores já enfrentaram em outras situações. Isso faz do *framework* um mecanismo extremamente confiável, uma vez que as soluções já foram previamente testadas (SCHMIDT e BUSCHMANN, 2003).

Frameworks são investimentos a longo prazo. Os benefícios adquiridos de um *framework* não são necessariamente imediatos, pois os projetistas precisam de mais tempo para desenvolver um *framework* do que uma biblioteca de classes convencional. Além disso,

os usuários/clientes também necessitam de um tempo maior para aprender a utilizar essa tecnologia (TALIGENT, 1997).

O surgimento da Programação Orientada a Aspectos em 1997 (KICZALES *et al*., 1997) também impulsionou o surgimento de *Frameworks* Orientados a Aspectos (CAMARGO e MASIERO, 2005; CAMARGO, 2006). Esse tipo de *framework* difere dos *frameworks* convencionais no sentido de que não geram (ou instanciam) uma determinada aplicação. Eles devem ser "acoplados" a uma aplicação, pois encapsulam a parte básica e genérica de apenas um determinado interesse, como, por exemplo: Persistência, Controle de Acesso e Autenticação.

No processo de desenvolvimento de software, mudanças ocorrem freqüentemente. Por isso, é importante que haja algum mecanismo para controlar tais mudanças. À medida que o software evolui, muitas versões diferentes são criadas. Essas versões incorporam propostas de mudanças, correções e adaptações para diferentes tipos de tecnologias. Portanto, existe a necessidade de manter o controle das mudanças que foram implementadas e de como essas mudanças foram incorporadas ao software. O gerenciamento de configuração de software *(Software Configuration Management – SCM)* tem como finalidade gerenciar todas as fases do ciclo de vida de um software, estabelecendo regras formais para identificar e controlar sistematicamente as mudanças realizadas, sendo também um dos requisitos para a garantia da certificação de qualidade (PRESSMAN, 2001; SOMMERVILLE, 2004).

Ressalta-se que uma das principais atividades do SCM é o gerenciamento de versões. O gerenciamento de versões faz parte das atividades de apoio ao desenvolvimento de software e tem como principal objetivo garantir a qualidade do software desenvolvido. Isso não é diferente no contexto de *frameworks.* A problemática é maior nesse contexto, uma vez que há necessidade de controlar não somente as versões do *framework*, como também a dos sistemas

vinculados aos *frameworks*[1]. No último caso é necessário saber a partir de qual versão do *framework* um determinado sistema foi criado pois, se houver mudanças no *framework* o sistema gerado pode não fornecer mais o comportamento desejado e se houver necessidade de evoluir o sistema, obtendo novas funcionalidades a partir do *framework*, é necessário que isso seja feito, em geral, a partir da versão do *framework* utilizada inicialmente a fim de minimizar os problemas que possam vir a ocorrer.

Para apoiar o gerenciamento de versões de maneira sistemática é indispensável a utilização de ferramentas específicas para isso. Essas ferramentas são essenciais na área de SCM, pois visam a facilitar o acesso às informações, garantir a integridade dos dados, controlar versões dos artefatos e acompanhar as requisições de mudanças.

Diversas ferramentas de gerenciamento de versões foram encontradas na literatura, porém nenhuma delas é para uso específico de controle de versões de *frameworks*: *Source Code Control System* – SCCS (BOLINGER E BRONSON, 1995), *Revision Control System* – RCS (TICHY, 1985), *Concurrent Versions System* – CVS (CEDERQVIST, 1993), *SubVersion* (COLLINS-SUSSMAN; FITZPATRICK; PILATO, 2004).

A partir disso, observou-se a importância do desenvolvimento de uma ferramenta de controle de versões específica para controlar o cadastro das versões dos *frameworks* e dos sistemas gerados, utilizando padrões e técnicas existentes, a fim de colaborar para o aumento da garantia de qualidade do software sob essa perspectiva.

[1] Optou-se por utilizar a palavra "vinculada" para generalizar a forma de relacionamento entre um framework e uma aplicação. Na literatura, o termo mais utilizado para designar o relacionamento entre um framework e uma aplicação é "instanciação", o que implica que um framework "instancia" ou "gera" uma determinada aplicação. Entretanto, no caso de frameworks orientados a aspectos isso não ocorre, pois eles são acoplados a uma determinada aplicação. Dessa forma optou-se por dizer que uma aplicação está

Motivação

Com a crescente demanda pelo desenvolvimento de software com maior rapidez, qualidade e eficiência, a técnica de reúso de software propiciada pelo *framework* tem atraído a atenção de muitos desenvolvedores, tornando-se uma prática bastante interessante. Entretanto, ainda há carência em vários aspectos tanto relacionados ao seu desenvolvimento quanto ao seu gerenciamento. Assim, observou-se a ausência na literatura de uma ferramenta computacional específica para auxiliar todo esse processo. Pois, apesar de *framework* ser considerado também um software, há peculiaridades que devem ser consideradas no controle de versões, como a necessidade de controlar as versões do *framework*, bem como a dos sistemas vinculados a ele.

Objetivo

O principal objetivo desse trabalho é desenvolver uma ferramenta de apoio ao controle de versões de *frameworks* visando a criar um repositório dos dados de cada versão. É necessário manter um registro dos requisitos dessas versões com os dados dos respectivos sistemas gerados a partir da instanciação de cada versão do *framework*. Além disso, a ferramenta deve armazenar o código fonte de cada versão do *framework*. Um registro dos requisitos não encontrados no *framework* durante o seu uso também será armazenado, a fim de apoiar a decisão sobre evoluir ou não o *framework*, com o apoio do PREF (Processo de Evolução de *Frameworks* de Aplicação) (CAGNIN, 2005). Todo esse controle deve ser feito constantemente para evitar que o sistema possa vir a não fornecer o comportamento desejado quando da utilização de versões do *framework* em que o sistema foi inicialmente gerado.

---

vinculada ao framework para servir tanto para frameworks convencionais quanto para frameworks

Organização da Monografia

Este trabalho está organizado em 3 capítulos, conforme descrito a seguir:

No Capítulo 1 apresenta-se a revisão bibliográfica necessária para o entendimento e a realização deste trabalho.

No Capítulo 2 é apresentada uma descrição detalhada do projeto, mostrando as principais funcionalidades, o modelo conceitual e arquitetônico, a implementação e um estudo de caso para validar e exemplificar o uso da ferramenta proposta. Serão abordadas também as dificuldades e limitações encontradas durante o desenvolvimento deste trabalho.

No Capítulo 3 apresentam-se as conclusões, ressaltando os resultados obtidos e as contribuições, além de sugestões de possíveis trabalhos futuros.

No Apêndice A apresenta-se o Documento de Requisitos. No Apêndice B apresenta-se a descrição dos casos de uso de todos os atores do sistema (usuários). No Apêndice C apresenta-se o script da criação da base de dados da ferramenta. Finalmente, no Apêndice D apresenta-se o Manual do Usuário da Ferramenta.

---

orientados a aspectos.

1. REVISÃO BIBLIOGRÁFICA

1.1. Considerações Iniciais

Neste capítulo são abordados os principais aspectos relacionados a *frameworks*, gerência de configuração de software, gerência de configuração sob a perspectiva de *frameworks* e ferramentas computacionais para apoiar esse processo. Todas essas informações constituem o embasamento teórico necessário para a compreensão e o desenvolvimento do trabalho em questão. Na Seção 1.2 são apresentadas as diversas definições para *frameworks* encontradas na literatura, descrição de suas principais características e os benefícios obtidos com o uso dessa tecnologia no desenvolvimento de aplicações em domínios específicos. Na Seção 1.2.1 apresenta-se a programação orientada a aspectos e também os *frameworks* orientados a aspectos. Na Seção 1.3 são abordados os diversos aspectos inerentes ao SCM. É relatada a necessidade da utilização desta abordagem na fase de desenvolvimento e no decorrer do ciclo de vida de um software e a sua contribuição para a área de Engenharia de Software. Ainda nessa seção, são apresentadas as principais ferramentas de apoio ao gerenciamento de versões. Na Seção 1.3.2 é discutido o gerenciamento de configuração sob a perspectiva de *frameworks*, destacando a problemática de controle de versão no desenvolvimento de aplicação baseado em *framework*. Na Seção 1.3.3 relata-se a carência de ferramentas computacionais específicas para apoiar esse processo. Apresenta-se também uma ferramenta que foi desenvolvida para controlar as versões dos sistemas gerados por um determinado *framework*. Finalmente, na Seção 1.4 são apresentadas as conclusões finais deste capítulo.

1.2. Conceitos de *Frameworks*

A utilização de *frameworks* no desenvolvimento de aplicações em domínios específicos vem crescendo significativamente nos últimos anos, visando o reúso de software e, conseqüentemente, à melhoria da produtividade, qualidade e manutenibilidade (TALIGENT, 1997; BRAGA, 2003).

O principal objetivo de um *framework* é facilitar o reúso, sendo também uma das metas do engenheiro de software. Reúso não somente de código de programas, mas também de análise e projeto (ROBERT e JOHNSON, 1996; BOSCH *et al.,* 1997; BRAGA, 2003). Reutilizar software não é uma tarefa trivial, uma vez que a maioria dos esforços resulta apenas na reutilização de pequenos componentes. No entanto, com o avanço do paradigma orientado a objetos, a tecnologia de reúso de grandes componentes tornou-se possível (BOSCH *et al.,* 1997). Utilizando a abordagem orientada a objetos, o desenvolvedor pode especializar, instanciar ou reutilizar as classes disponíveis, respeitando os relacionamentos já definidos.

Os *frameworks* são descritos em termo dos principais conceitos da tecnologia orientada a objetos *(object oriented technology – OOT),* como classes, objetos, herança e polimorfismo (TALIGENT, 1997), fornecendo a base necessária para explorar completamente as potencialidades dessa tecnologia.

O maior benefício de utilizar *frameworks* é que eles permitem um nível maior de reúso de código e projeto do que aquele que é alcançado com outras abordagens de projeto (TALIGENT, 1998; BRUGALI, 2000 apud CAGNIN, 2005).

Frameworks são tecnologias promissoras para as empresas de software. Atualmente essa técnica de reúso têm gerado muita discussão e atraído a atenção de muitos pesquisadores e engenheiros de softwares. Diversas definições sobre essa tecnologia são encontradas na

literatura. Uma definição amplamente aceita vem de Johnson e Foote (1988): um *framework* é um conjunto de classes que engloba um projeto abstrato de soluções para um grupo de problemas relacionados. Outra perspectiva para *frameworks* é como uma estrutura pré-fabricada ou modelo de um programa funcionando (TALIGENT, 1993) ou, ainda, como um programa parcial para um domínio de problemas, que desenvolvedores podem estender com interfaces externas para construir aplicações específicas (BROECKE e COPLIEN, 1997).

Adicionalmente, um *framework* é definido como uma aplicação "semi-completa" que programadores podem personalizar para gerar aplicações prontas, apenas estendendo componentes reutilizáveis presentes no *framework* (SCHMIDT e BUSCHMANN, 2003). *Frameworks* são projetos reutilizáveis de todo ou parte de um sistema de software, representados por uma série de classes abstratas e pela forma com que essas classes colaboram entre si, objetivando a realização de uma determinada tarefa (ROBERT e JOHNSON, 1996; JOHNSON, 1997).

Segundo Taligent (1997), o *framework* consiste de uma coleção de componentes de software pré-fabricados que desenvolvedores podem utilizar, estender ou adaptar para soluções computacionais específicas. *Frameworks* servem como base para que os desenvolvedores não iniciem do zero quando necessitarem de uma nova aplicação.

O *framework* pode ser definido também como uma arquitetura reutilizável, que pode ser especializada para produzir aplicações personalizadas (JOHNSON e FOOTE, 1988; FAYAD e SCHMIDT, 1997).

Frameworks fornecem um modo inovador de reutilizar projeto e código (TALIGENT, 1997) e são apropriados para domínios em que aplicações similares são construídas diversas vezes (FIORINI, 2001).

Taligent (1993) acredita que *frameworks,* que é a parte central deste novo ambiente operacional, é o avanço mais importante em OOT, porque *frameworks* provêem infra-

estrutura e interfaces flexíveis. Com *frameworks* bem projetados é muito mais fácil adicionar extensões para eliminar funcionalidades comuns, aumentar a produtividade, melhorar a manutenção e a confiança do software.

Framework é uma técnica poderosa que facilita o reúso de software, uma vez que uma grande variedade de diferentes aplicações podem ser obtidas a partir da sua instanciação, diminuindo drasticamente o tempo de desenvolvimento. No entanto, o processo de instanciação normalmente é muito complexo, o que pode desencorajar o seu uso (BRAGA e MASIERO, 2004).

Segundo Fayad e Schmidt (1997), os benefícios primários de um *framework* originam da modularidade, reusabilidade, extensibilidade e da inversão de controle.

Modularidade: *frameworks* aumentam a modularidade encapsulando detalhes da implementação por trás de interfaces estáveis. *Framework* modularizado ajuda a melhorar a qualidade do software identificando o impacto de projeto e implementação das mudanças e reduzindo os esforços exigidos para entender e manter o software existente.

Reusabilidade: as interfaces estáveis fornecidas pelos *frameworks* aumentam a reusabilidade definindo componentes genéricos que podem ser reaplicados para criar novas aplicações. *Frameworks* reusáveis podem ser adaptados facilmente às necessidades específicas de um domínio.

Extensibilidade: os *frameworks* são extensíveis pois provêem métodos explícitos e adaptáveis que permitem às aplicações estenderem suas interfaces.

Inversão de controle: é uma das características mais importantes de um *framework*. Os métodos definidos pelo usuário para especializá-lo são invocados dentro do próprio *framework*, ao invés de serem invocados do código da aplicação do usuário. O *framework* geralmente desempenha a mesma função do programa principal, gerenciando as atividades da

aplicação. Essa abordagem dá força ao *framework* para servir de esqueletos extensíveis de programação.

Os *frameworks* fornecem, simultaneamente, aos engenheiros de softwares, o modelo arquitetural com seus padrões e estruturas, e uma implementação dessas estruturas compostas por componentes genéricos (classes abstratas) que definem as interfaces de acoplamento e controle usadas pelo programa principal para definir o fluxo de controle da aplicação (TIBERTI, 2003). Isso é uma vantagem considerável em relação a outras tecnologias.

A idéia do *framework* é separar partes estáticas, que permanecem inalteradas, das partes que estão sujeitas a alterações. Seguindo esse princípio, existem as partes fixas *(frozen spots)* e as partes variáveis *(hot spots)* (SOUZA, 2004).

As partes fixas definem a arquitetura geral de uma aplicação, sendo comum ao domínio em que o *framework* é utilizado, e não sofrem alterações quando o *framework* é instanciado (BRAGA, 2003; SOUZA, 2004).

As partes variáveis são diferentes aplicações dentro de um mesmo domínio e que se diferenciam por uma ou mais partes variáveis. Elas descrevem as partes do *framework* de aplicação que são específicas de cada sistema. As partes variáveis são projetadas para serem adaptadas às necessidades da aplicação e para proverem uma funcionalidade especial (BRAGA, 2003).

Fayad e Schmidt (1997) classificam os *frameworks* em três grupos, de acordo com o seu escopo, ou seja, a área de abrangência de sua aplicação em um determinado domínio. São eles: *frameworks* de infra-estrutura do sistema, *frameworks* de integração *middleware* e *frameworks* de aplicação empresarial.

Os *frameworks* de infra-estrutura do sistema facilitam o desenvolvimento da infra-estrutura de sistemas adaptáveis e eficientes, tais como sistemas operacionais e sistemas de comunicação.

Os *frameworks* de integração *middleware* geralmente são utilizados para integrar aplicações e componentes distribuídos, permitindo a troca de informações entre sistemas heterogêneos. Além disso, esses tipos de *frameworks* aumentam a capacidade dos desenvolvedores na modularização, reutilização e extensão da sua infra-estrutura de software para funcionar de forma integrada em ambientes distribuídos.

Os *frameworks* de aplicação empresarial destinam-se a domínios mais abrangentes e são primordiais para as atividades de negócios das empresas. Os custos desses *frameworks* são maiores, no entanto, podem dar um retorno substancial do investimento realizado.

Em relação à técnica de extensão, existem três tipos de *frameworks* (FAYAD e SCHMIDT, 1997; JOHNSON, 1997): caixa branca, caixa cinza e caixa preta.

O *framework* caixa branca está fortemente ligado ao conceito de orientação a objetos, uma vez que a sua instanciação ocorre por meio de herança e ligação dinâmica. Consiste de diversas classes incompletas, adaptadas às particularidades de uma aplicação específica com a adição de atributos e métodos às subclasses de uma ou mais classes do *framework*. Para tanto, os usuários/desenvolvedores precisam entender o projeto e a implementação do *framework* para utilizá-lo.

O *framework* caixa preta, em contraste ao *framework* caixa branca, oferece um conjunto de classes pré-definidas, fornecendo o comportamento específico da aplicação. Com base nesse conjunto, o usuário seleciona as melhores alternativas para modelar a aplicação. Desse modo, o usuário precisará apenas entender a interface, o que torna mais fácil de usar e estender, porém, mais difícil de se desenvolver.

O *framework* caixa cinza busca incorporar as principais características existentes nos *frameworks* caixa branca e caixa preta, sendo considerado um meio termo entre ambos. Possui bastante flexibilidade, extensibilidade e a habilidade de ocultar as informações desnecessárias

para desenvolvedores da aplicação. Por estas razões, a tendência no momento é pelo desenvolvimento desse tipo de *framework*.

O desenvolvimento de um *framework* é um pouco diferente do desenvolvimento de um software tradicional. A principal diferença é que um *framework* tem que cobrir os conceitos relevantes a um domínio, enquanto o software tradicional cobre apenas os conceitos descritos nos seus requisitos (BOSCH *et al.,* 1997). Tipicamente, um *framework* é desenvolvido por especialistas que têm um amplo conhecimento no domínio da aplicação e largas experiências em projetos de softwares.

Frameworks são investimentos a longo prazo. Os benefícios adquiridos de um *framework* não são necessariamente imediatos, pois os projetistas precisam de mais tempo para desenvolver um *framework* do que uma biblioteca de classes convencional. Os usuários/clientes também precisam de um tempo maior para aprender a utilizar essa tecnologia (TALIGENT, 1997).

Os *frameworks* fornecem a infra-estrutura e diretrizes arquiteturais de um produto de software. Como a maioria das funcionalidades já estão implementadas no *framework*, desenvolver aplicações a partir da sua instanciação torna-se mais rápido e eficiente, reduzindo o tempo de codificação, teste e depuração (BRAGA, 2003).

A melhor maneira de desenvolver aplicações a partir de um *framework* seria completar ou modificar procedimentos e estrutura de dados nele presente. Porém, um *framework* geralmente não contém a descrição completa de um domínio em particular, o que leva à criação de novas classes (GROTT, 2003a; GROTT; HUGO; SOUSA, 2003b).

A manutenção de *frameworks* não é tão simples já que o mantenedor tem que ter um conhecimento profundo da estrutura do *framework* e dos relacionamentos entre suas classes (FAYAD e SCHMIDT, 1997; BRAGA, 2003). Além disso, as aplicações geradas a partir da

instanciação do *framework* necessitam passar pelo mesmo processo de manutenção para haver uma compatibilidade com o *framework* modificado.

1.2.1. Programação Orientada a Aspectos e *Frameworks* Orientados a Aspectos

A Programação Orientada a Aspectos (POA) (KICZALES, 1996; KICZALES *et al*., 1997) e a linguagem AspectJ (KICZALES *et al*., 2001) surgiram no final da década de 90 como uma importante contribuição para modularizar interesses transversais, que até então ficavam misturados e espalhados pelo código orientado a objetos, sem que fosse possível organizá-los em módulos independentes. A POA fornece novas abstrações que contribuem para a separação de interesses (*separation of concerns*) – um objetivo antigo da Engenharia de Software que visa a separar os interesses encontrados no código-fonte de um sistema (DIJKSTRA, 1976 apud CAMARGO, 2006). Com a POA é possível implementar separadamente os interesses-base e os interesses transversais, o que até então era difícil somente com orientação a objetos. Os interesses-base referem-se à funcionalidade principal do sistema e os transversais referem-se a restrições globais e a requisitos não-funcionais, como, por exemplo: persistência, distribuição, autenticação, controle de acesso e concorrência.

A partir desse avanço, outras linguagens de programação orientadas a aspectos surgiram e iniciou-se um processo de revisão das técnicas, métodos e processos usados até então para adaptá-las a essa nova forma de "pensar" o software e modularizá-lo, tais como técnicas de elicitação e modelagem de requisitos, linguagens de modelagem para projeto, técnicas de teste e uso de aspectos para implementar variabilidades em linhas de produto de software.

Como a implementação de interesses transversais em unidades sintáticas abstratas também é possível com a POA, vários pesquisadores iniciaram pesquisas com o objetivo de implementar esses interesses de forma genérica para investigar seu reúso em outros contextos, criando assim, "*frameworks* orientados a aspectos"[2] que encapsulam um único interesse transversal (CONSTANTINIDES *et al*., 2000; VANHAUTE *et al*., 2001; PINTO *et al*, 2002; SOARES *et al*., 2002; RASHID e CHITCHYAN , 2003; HANENBERG *et al*., 2004; HUANG *et al*., 2004; SHAH e HILL, 2004; COUTO *et al*., 2005).

Do ponto de vista da estrutura, um FOA (*Framework* Orientado a Aspectos) é um conjunto formado por unidades básicas de programação orientada a objetos (OO) (classes), cuja presença não é obrigatória, e unidades básicas de programação orientada a aspectos (OA) (aspectos). A não obrigatoriedade de classes significa que um FOA pode ser composto apenas por aspectos, e embora isso não seja comum, é possível que ocorra em situações especiais. Esse conjunto representa o projeto abstrato de soluções para uma família de problemas relacionados.

Do ponto de vista do propósito, assim como os FOO (*Framework* Orientado a Objetos) (FAYAD *et al*, 1999), um FOA pode ser definido como um sistema "semi-completo" e reutilizável que pode ser instanciado por um desenvolvedor de aplicações. A arquitetura de um FOA possui uma parte fixa e uma parte variável, sendo que esta deve ser adaptada para que o *framework* seja acoplado a uma aplicação já existente ou produza uma nova aplicação. A adaptação geralmente envolve a concretização de um mecanismo de composição abstrato, no caso dos aspectos, e de classes e métodos abstratos, no caso das classes.

---

[2] Deste ponto em diante, o termo "framework orientado a aspectos" será usado para designar uma implementação genérica de um único interesse transversal, até que uma definição mais precisa seja apresentada no Capítulo 2.

Quanto à natureza, há dois tipos de FOAs: *Frameworks* Transversais e *Frameworks* de Aplicação Orientados a Aspectos. Os dois tipos são discutidos a seguir, dando-se ênfase aos *frameworks* transversais por serem o foco deste trabalho.

Um ***Framework* Transversal (*Crosscuting Framework*) (FT)** é um FOA que possui mecanismos de composição abstratos e variabilidades correspondentes a um único interesse transversal, como por exemplo: persistência, distribuição, segurança e regras de negócio. Como um interesse pode ser particionado em sub-interesses, fica a critério do projetista implementar o sub-interesse com aspectos. Caso isso seja feito, também é sua responsabilidade determinar o nível de granularidade desses interesses e se cada um será projetado ou não na forma de um FT. Uma outra característica desse tipo de *framework* é que ele só pode ser utilizado se for acoplado a algum código-base existente, isto é, seu reúso não produz uma aplicação. Sendo assim, seu processo de reúso possui duas etapas "semanticamente" distintas: instanciação e composição.

A instanciação é o processo convencional de reúso dos FOOs tradicionais e consiste em especializar o código que foi especialmente projetado para isso. É o processo pelo qual implementam-se os ganchos do *framework*, escolhe-se alguma funcionalidade alternativa ou implementa-se uma nova. Isso é feito normalmente sobrepondo-se métodos que retornam valores da aplicação específica.

A etapa de composição por sua vez, consiste em duas atividades: identificação dos pontos de junção e fornecimento de regras de composição. A primeira atividade consiste em identificar no código-base o(s) ponto(s) de junção adequado(s) ao acoplamento e deve ser feita com base nas "alternativas de composição" disponíveis no *framework*. É interessante que os FTs sejam projetados com alternativas de composição, principalmente aqueles que necessitam de dados da aplicação em seu processamento. Essas alternativas aumentam as chances de acoplamento com códigos-base já existentes, além de diminuir a complexidade das

regras de composição que precisam ser fornecidas. A necessidade de alternativas de composição torna-se mais evidente quando o *framework* deve ser acoplado a um código-base já existente. No caso de um novo desenvolvimento, o código-base já pode ser projetado com vistas ao acoplamento que será feito. Contudo, esse "desenvolvimento orientado às alternativas de composição" pode tornar o código-base confuso e difícil de manter, pois pode ser que pontos de junção fictícios tenham que ser criados apenas para o acoplamento.

A segunda atividade da etapa de composição consiste em fornecer regras que unem as variabilidades escolhidas/implementadas do *framework* com o código-base, e para isso tarefas orientadas a aspectos devem ser realizadas como, por exemplo, a concretização de um mecanismo de composição abstrato ou a utilização de declarações inter-tipo, no caso da linguagem AspectJ. Em alguns casos, a etapa de composição depende de informações que são determinadas na etapa de instanciação, o que determina uma ordem de realização: primeiro a instanciação e depois a composição. Mas também pode ocorrer de não haver essa dependência, fazendo com que essas etapas possam ser feitas em qualquer ordem ou em paralelo.

Embora essas duas etapas sejam semanticamente distintas, sua separação "física" pode não existir durante o reúso, pois isso depende do projeto do *framework* e da linguagem orientada a aspectos utilizada. Visto que algumas abordagens OA tendem a separar as regras de composição do comportamento transversal, infere-se que o projeto de um *framework* deste tipo deve ser elaborado com o objetivo de separar as duas etapas tanto quanto possível para que a separação semântica continue existindo fisicamente. Contudo, como AspectJ é uma abordagem que mantém as regras de composição no mesmo módulo do comportamento transversal, um bom projeto precisa ser elaborado para separar fisicamente essas duas etapas.

Quando se utiliza FTs não há inversão de controle do ponto de vista da aplicação, pois é ela que determina o fluxo principal. Porém, do ponto de vista interno do *framework* isso

acontece, pois, como qualquer *framework* orientado a objetos, ele possui código genérico que chama métodos abstratos implementados na aplicação específica. Em outras palavras, não é responsabilidade do engenheiro de aplicações chamar métodos do *framework*. Pode-se dizer assim que o fluxo de controle principal é da aplicação, enquanto que nos FTs o fluxo de controle é deles. Essa é outra característica diferente das bibliotecas de classes, pois quando elas são utilizadas, o fluxo de controle principal é sempre da aplicação. Nesse sentido, os FTs são similares aos *framelets* (PREE, 1999), pois não assumem o controle da aplicação, têm interface de composição simples e bem definida, e geralmente, mas não necessariamente, têm um número pequeno de unidades de programação (classes e aspectos).

Neste trabalho optou-se por utilizar o termo "*framework* orientado a aspectos" no mesmo sentido de "*frameworks* tranversais".

1.3. Gerência de Configuração de Software

Os softwares são entidades inerentemente passíveis de mudança devido a eventuais problemas encontrados na fase de desenvolvimento, necessidade de adicionar novas funcionalidades ou inserir melhorias. É nesse contexto que o SCM é tão importante, tornando-se imprescindível para a obtenção de um software de melhor qualidade.

De acordo com Sommerville (2004), o SCM consiste no desenvolvimento e na aplicação de padrões e procedimentos para gerenciar o software ao longo do tempo. À medida que o software evolui, muitas versões diferentes são criadas. Essas versões incorporam propostas de mudanças, correções e adaptações para diferentes tipos de tecnologias. Portanto, existe a necessidade de manter o controle das mudanças que foram implementadas e de como essas mudanças foram incorporadas ao software.

Sendo assim, pode-se concluir que os softwares são altamente mutáveis, principalmente em se tratando de softwares organizacionais, pois as necessidades e os requisitos organizacionais requerem que mudanças sejam feitas no software a qualquer momento. Um dos objetivos da Engenharia de Software é justamente facilitar a maneira com que as modificações são adaptadas e minimizar os esforços despendidos na implementação dessas mudanças (PRESSMAN, 2001).

O SCM pode ser visto como uma abordagem utilizada para gerenciar todas as fases do ciclo de vida de um software, estabelecendo regras formais para identificar e controlar as mudanças realizadas de modo sistemático, mantendo a estabilidade na evolução do software. Na visão de Tichy (1988), o SCM é uma área da Engenharia de Software responsável pelo controle da evolução de sistemas mais complexos. Formalmente, essa abordagem permite ao desenvolvedor manter a evolução de sistemas de software grandes e complexos sob controle.

Segundo Pressman (2001), o SCM é uma atividade abrangente considerada como um dos requisitos para a garantia de qualidade de software, aplicada em todo processo de Engenharia de Software. Essas atividades incluem: identificar a mudança, controlar a mudança, garantir que a mudança seja implementada corretamente e relatar a mudança a outras pessoas interessadas. O SCM fornece uma metodologia consistente, flexível e disciplinada para gerenciar as evoluções das mudanças e para garantir a integridade e a rastreabilidade dos artefatos alterados.

O SCM é uma disciplina de natureza técnica que veio cobrir as lacunas sobre como, quando e o que gerenciar em um processo de desenvolvimento e evolução de um produto de software. A finalidade do SCM é gerenciar as modificações realizadas no ciclo de vida do software, dando ênfase na fase de desenvolvimento (MACHADO, 2002). Sua importância tem sido amplamente reconhecida e refletida, em particular, no padrão *Capability Maturity Model* (CMM), desenvolvido pelo *Software Engineering Institute* (SEI) (CONRADI e

WESTFECHTEL, 1998), com o intuito de definir níveis de maturidade para avaliar e melhorar o processo de desenvolvimento de software nas organizações (WESTFECHTEL, 2003).

O SCM é um dos meios para alcançar um software de melhor qualidade, enquanto minimiza o processo de desenvolvimento e os esforços necessários na manutenção do software. Para um efetivo SCM é preciso seguir uma série de políticas, processos e padrões bem definidos e institucionalizados. Dentro de uma organização, esses padrões devem ser publicados em um manual de SCM ou como parte de um manual de qualidade. Não importa qual padrão seja considerado ponto de partida, uma vez que todos eles apresentam processos comparáveis e as organizações devem definir e seguir padrões formais de SCM para a certificação de qualidade (SOMMERVILLE, 2004).

Segundo Silva (2005), a falta de controle no desenvolvimento e ciclo de vida de um software pode resultar em inúmeros problemas, como: problemas de interface na integração de módulos; desaparecimento de funcionalidades acrescentadas após as modificações; surgimento de erros corrigidos anteriormente; perda dos arquivos fonte e dificuldades em identificar a última versão do produto e, como conseqüência, levar a altos custos de desenvolvimento, baixa qualidade do software e a insatisfação do cliente ou usuário final.

Rueda e Carlos (2003) relatam ainda que muitos problemas antes difíceis de se contornar como, versões de software, arquivos fontes duplicados, acompanhamento do processo, entre outros, são facilmente contornados com a utilização do SCM. Dessa forma, o SCM deve, naturalmente, ser parte integrante de qualquer atividade de desenvolvimento de software. Práticas de SCM são especialmente importantes em esforços nos quais muitos desenvolvedores estão produzindo cooperativamente e usando vários artefatos compartilhados. SCM é importante até mesmo quando houver apenas um desenvolvedor ou uma pequena equipe de desenvolvedores.

Muitas pessoas tendem a relacionar a manutenção de software com o SCM. No entanto, Pressman (2001) destaca a importância de diferenciar essas duas abordagens. A manutenção de software caracteriza-se por um conjunto de processos de Engenharia de Software aplicados após a entrega efetiva do software ao cliente. O SCM por sua vez é uma série de atividades de controle e rastreamento introduzido desde o início de um projeto de desenvolvimento até o software ser retirado de operação.

O SCM é composto basicamente por cinco atividades:

1. **Planejamento de gerenciamento de configuração:** no processo de SCM é preciso que haja planejamento. O plano de SCM envolve padrões e procedimentos que devem ser aplicados ou adaptados para toda a organização (SOMMERVILLE, 2004). Para isso, é preciso definir os responsáveis pela entrega de cada documento ou componente de software para o gerenciamento da garantia de qualidade e configuração, e também os responsáveis pela revisão dos documentos (SILVA, 2005).
    - **Identificação dos itens de configuração:** no decorrer do processo de planejamento de SCM são identificados os itens a serem controlados, ou seja, os itens de configuração, que são todos os elementos necessários para o desenvolvimento e a geração dos produtos de software. Contudo, todos os documentos que podem ser úteis para uma manutenção futura devem ser controlados (SOUZA, 2004).
    - **Repositório de configuração:** para armazenar todas as informações relevantes relacionadas às configurações, é necessário definir o repositório de configuração. O repositório de configuração auxilia na avaliação do impacto das mudanças provocado nos sistemas e fornece informações gerenciais à respeito do processo de SCM. Além disso, deve responder a uma infinidade de consultas referente às configurações do sistema (SOMMERVILLE, 2004).

2. **Gerenciamento de mudanças:** trata de todas as mudanças requeridas em um sistema, fornecendo um serviço similar ao oferecido pelo sistema de gerenciamento de versões. O controle de mudanças combina procedimentos humanos e ferramentas automatizadas para gerar um mecanismo de controle das mudanças (PRESSMAN, 2001). Dois objetivos principais devem ser alcançados com o gerenciamento de mudanças (SILVA, 2005):
   - o fornecimento de um processo em que as solicitações de mudanças são priorizadas e as decisões para implementar ou rejeitar são tomadas e;
   - a possibilidade de consultar todas as solicitações ativas e implementadas e rastrear todas as mudanças realizadas.
3. **Gerenciamento de versões:** com o aumento inevitável de várias versões de software (devido às modificações realizadas), surge a necessidade de gerenciar as novas versões criadas (TICHY, 1985). O gerenciamento de versões é o meio pelo qual o SCM se utiliza para controlar de forma consistente e disciplinada as modificações realizadas, sendo considerada uma das principais atividades no contexto de SCM. O gerenciamento de versões combina procedimentos e ferramentas para monitorar as diversas versões de itens de configuração geradas durante o processo de Engenharia de Software (PRESSMAN, 2001). Somente a equipe de SCM poderá gerar e modificar as novas versões, a fim de garantir a consistência do repositório de configuração. Cabe aos gerentes de configuração controlar as diferenças entre as versões de software, a fim de assegurar que as versões derivem de forma correta e controlada. Para haver um controle sistemático das versões, o processo de gerenciamento de versão tem que definir de forma atômica como identificar cada versão do sistema. Segundo Sommerville (2004) existem três técnicas básicas para a identificação de versões de sistema:

- Numeração de versões: é o formato em que cada versão tem um número de identificação. Essa é forma de identificação mais utilizada.
- Identificação baseada em atributos: os componentes são identificados pela combinação de nomes (que não é único em todas as versões) e um conjunto de atributos associados.
- Identificação orientada a mudanças: a versão do sistema é identificada associando um nome com as mudanças implementadas no software.

4. **Auditoria de Configuração:** procedimento que determina se um item de configuração está de acordo com o documento de configuração, dando a garantia que as modificações foram implementadas adequadamente e que a qualidade do software foi mantida nas implementações dessas mudanças.
5. **Relato do status de configuração:** relatório com todas as informações relevantes das mudanças realizadas. É essencialmente importante em grandes projetos de desenvolvimento de software para organizar, relatar e melhorar a comunicação entre a equipe de profissionais.

É importante ressaltar que a adoção do SCM não visa a burocratizar o serviço, nem aumentar o nível de complexidade, mas sim, facilitar e agilizar o processo de Engenharia de Software. Qualquer que seja a resistência nesse sentido deve ser trabalhada, a fim de que os diversos benefícios decorrentes da prática de SCM possam ser assimilados pelos profissionais envolvidos no processo.

1.3.1. Ferramentas de Apoio ao Gerenciamento de Versões

A qualidade de um projeto de software é similar à qualidade dos processos adotados em todas as fases do seu ciclo de vida. O gerenciamento de versões é parte fundamental no

processo de SCM, permitindo o trabalho paralelo de maneira consistente e uniforme. Sem uma ferramenta de apoio ou metodologia adequada, esse gerenciamento torna-se extremamente complexo e ineficaz.

As ferramentas de apoio ao gerenciamento de versões são essenciais no contexto de SCM visando a facilitar o acesso às informações, garantir a integridade dos dados, controlar versões dos artefatos e acompanhar as requisições de mudanças.

A seguir serão apresentadas algumas das principais ferramentas de apoio ao SCM disponíveis no mercado, descrevendo sua aplicabilidade e contribuição para o SCM, incluindo a RCS *(Revision Control System)* e a CVS *(Concurrent Control System),* consideradas as mais importantes e mais difundidas no processo de desenvolvimento de software, principalmente em projetos de software livre (FUTRELL; SHAFER; SHAFER, 2002; SILVA, 2002; BAR e FOGEL, 2003; CAETANO, 2005). Nesta seção, serão abordadas somente as ferramentas de apoio ao gerenciamento de versões *open-source/free* devido ao baixo custo, uma das razões que motivou a realização deste trabalho.

- ***Source Code Control System* (SCCS)**

A SCCS é uma ferramenta bem antiga e tem como meta principal controlar as mudanças que ocorrem durante o desenvolvimento de código fonte. Foi desenvolvida por Mac Rochkind em 1972 (BOLINGER e BRONSON, 1995). Tem grande valia quando se deseja manter, de forma confiável, várias versões de um arquivo, alterar um arquivo freqüentemente e recuperar ou visualizar versões anteriores (BAR e FOGEL, 2003).

A maioria das ferramentas de gerenciamento de versões usa o conceito de deltas para armazenar as versões com o intuito de economizar espaço de armazenamento. Essa técnica é bastante eficiente quando utilizada em arquivos do tipo texto.

O Delta é definido por Tichy (1985) como uma série de comandos de edição para transformar uma cadeia de caracteres *(string)* em outra. Um delta é formado pela seqüência de comandos que, quando aplicada a uma determinada versão, gera outra versão do arquivo. Portanto, ele representa as diferenças entre duas versões. O SCCS usa deltas intercalados para as versões de um arquivo. De modo geral, esses arquivos são particionados em blocos de linhas e cada bloco tem um cabeçalho para indicar a quais revisões ele pertence.

Existem várias abordagens para a geração de deltas entre arquivos do tipo texto. Entre elas, pode-se citar o algoritmo proposto por Tichy (1985):

- **Delta Positivo:** a primeira versão de um sistema é armazenada integralmente e, as posteriores, na forma de delta;
- **Delta Negativo ou Delta Reverso:** a versão mais recente de um sistema é armazenada integralmente e as anteriores na forma de delta.

Com base nessa abordagem, a ferramenta SCCS armazena somente a primeira versão integralmente, ou seja, usa a técnica de delta positivo. Assim, a primeira versão poderá ser recuperada rapidamente (CONRADI e WESTFECHTEL, 1998; BAR e FOGEL, 2003).

Na Figura 1.1 apresenta-se uma árvore de revisão com um ramo lateral representado pelo retângulo. Os triângulos direcionados à esquerda e à direita representam deltas negativos e positivos, respectivamente.

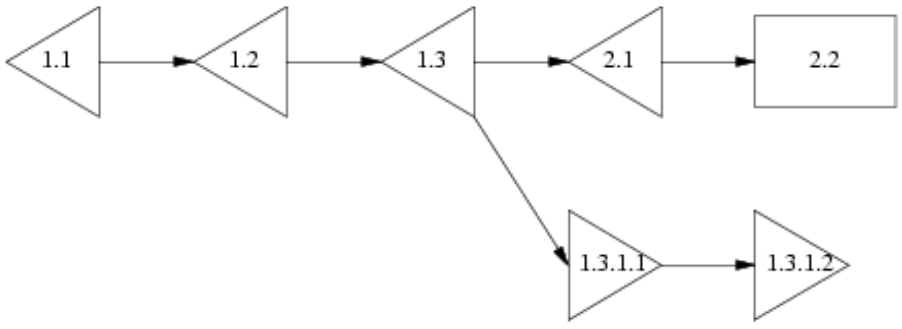


Figura 1.1 – Árvore de revisão com deltas negativos e positivos (TICHY, 1985)

Uma das limitações do SCCS é que ele só suporta arquivos texto, além de não atender algumas necessidades ou facilidades disponíveis atualmente como permitir que grupos de desenvolvedores possam trabalhar simultaneamente em um mesmo projeto, restringindo dessa forma um pouco a sua aplicabilidade.

- ***Revision Control System* (RCS)**

A RCS é uma ferramenta que auxilia no gerenciamento de versões de arquivos gerados durante o processo de desenvolvimento de um software (TICHY, 1985). RCS gerencia revisões de documentos textos, em particular códigos fontes, documentação e dados de testes. Além disso, a ferramenta automatiza o armazenamento e a recuperação de informações, identifica revisões e fornece mecanismos de seleção para composição de configurações (SOARES, 2000). A RCS serve como base para muitas ferramentas avançadas de controle de versões, incluindo a CVS.

A RCS, apesar de ser considerada uma ferramenta amigável, baseia-se em um conjunto de comandos UNIX, o que demanda treinamento dos usuários. Por outro lado, recursos importantes não presentes na ferramenta SCCS foram incorporados, como por exemplo, uma maior otimização na escrita e na recuperação de um arquivo (LACERDA; BARTH; GOMI, 2001).

A principal funcionalidade da RCS é gerenciar os grupos de revisão, formada por um conjunto de documentos no formato texto. As revisões são organizadas em árvore “ancestral” (sistema de cópia de arquivos do mais velho para o mais novo). A revisão inicial é a raiz da árvore e as conexões entre elas indicam de qual revisão uma outra foi gerada (TICHY, 1985; SOARES, 2000; BAR e FOGEL, 2003). Essa ferramenta não permite que desenvolvedores acessem simultaneamente a mesma revisão de um arquivo.

Para economizar espaços de armazenamento, usa o conceito de deltas. Na RCS um delta é baseado em linhas de textos, o que indica que os únicos comandos de edição permitidos são “inserção” e “exclusão” de linhas. Se um caracter em uma determinada linha for alterado, considera-se que a linha inteira foi alterada (TICHY, 1985; SILVA, 2005).

A forma de armazenamento da RCS é diferente da SCCS. A RCS usa a técnica de delta negativo ou delta reverso, ou seja, armazena a versão mais recente integralmente e as anteriores por meio de deltas, permitindo que a versão mais recente seja recuperada rapidamente (TICHY, 1985; CONRADI e WESTFECHTEL, 1998).

A RCS é fácil de administrar e bastante eficiente em documentos mais simples (BAR e FOGEL, 2003). Por outro lado, tanto a RCS quanto a SCCS, conforme descrito anteriormente, não suportam arquivos binários. Além disso, ambas não permitem acesso ao repositório por meio da rede e não possibilitam trabalhar em paralelo.

- ***Concurrent Versions System*** **(CVS)**

O CVS (CEDERQVIST, 1993) foi originalmente desenvolvido por Dick Grune em 1986 e consistia de um conjunto de comandos UNIX. Em 1989 Brian Berliner projetou e codificou o CVS na linguagem de programação C, com o auxílio de Jeff Polk no desenvolvimento de suporte a *branches* (ramos) (BAR e FOGEL, 2003). Um ramo é um termo usado para organizar as versões, atribuindo um número para as alterações realizadas em um arquivo. Desse modo, cria-se um ramo partindo da revisão de um arquivo.

Conforme ilustrado na Figura 1.2, um ramo é criado a partir da versão 1.1, e assim sucessivamente. A criação de um ramo se faz necessária no caso de mudanças em paralelo realizadas no mesmo arquivo por diferentes desenvolvedores ou para indicar um caminho independente de desenvolvimento.

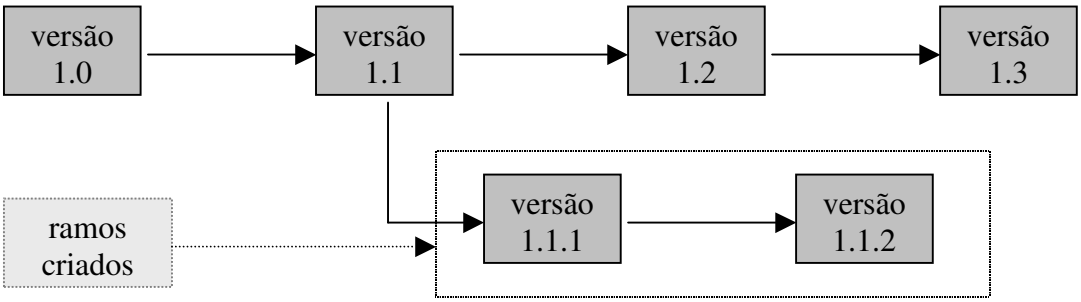


Figura 1.2 – Exemplo de ramos

O CVS surgiu baseado na RCS com alguns recursos adicionais, como a possibilidade de gerenciar arquivos binários, criação de uma hierarquia de diretórios dentro de um repositório, acesso a esse repositório por meio da rede e desenvolvimento paralelo. Tais recursos deram mais poder à ferramenta, difundindo mundialmente seu uso.

O CVS é uma ferramenta multi-plataforma bastante utilizada em ambientes de software, sobretudo em projetos de software livre, em virtude das suas características e da simplicidade de uso e configuração.

O CVS implementa as principais funções pertinentes ao processo de gerenciamento de versões. Sua finalidade é identificar e controlar as modificações nos arquivos de um projeto ao longo do tempo, garantido a integridade e a rastreabilidade das modificações por meio de um mecanismo automatizado (CAETANO, 2005).

O CVS possibilita a realização de modificações paralelas de forma eficiente e padronizada, especialmente em se tratando de equipes geograficamente dispersas.

Vários desenvolvedores podem trabalhar simultaneamente em um mesmo projeto, se encarregando de "mesclar" as versões e relatar os conflitos (BAR e FOGEL, 2003; SILVA, 2005).

Assim como outras ferramentas de apoio ao SCM, o CVS possui um repositório central para armazenar os arquivos sob o gerenciamento de versões (SOARES, 2000),

juntamente com o histórico de mudanças que permite o acompanhamento de todas as modificações realizadas ao longo do tempo. No repositório estão as informações necessárias para recriar qualquer revisão, além de indicar os responsáveis e as razões pelas quais as modificações foram efetuadas (SILVA, 2005).

A cada versão de um arquivo é atribuído um número de revisão (CAETANO, 2005). Qualquer revisão pode ser rastreada para fins de consulta, comparação ou efetuando junção com as demais revisões.

O CVS permite a criação de uma hierarquia de diretórios dentro de um único repositório, armazenando grandes projetos de software de forma bem organizada. Naturalmente, uma configuração pode ser constituída por versões de arquivos armazenados em diferentes diretórios (SILVA, 2002).

Ao contrário de outras ferramentas estudadas anteriormente, o CVS suporta tanto arquivos texto quanto binário, embora arquivos no formato texto sejam mais comuns devido a algumas facilidades adicionais, tais como a possibilidade de visualizar as diferenças entre as revisões e efetuar operação de junção de revisões sinalizando os conflitos em determinadas mudanças.

- ***Subversion***

A ferramenta *Subversion* busca aprimorar o CVS, contudo seguindo os mesmos princípios básicos. Pode ser considerado um provável substituto do CVS em projetos de software livre.

Karl Fogel e Bem Collins-Sussman iniciaram o processo de desenvolvimento da ferramenta em 2000, com o auxílio de outros desenvolvedores (COLLINS-SUSSMAN; FITZPATRICK; PILATO, 2004).

A *Subversion* gerencia arquivos e diretórios ao longo do tempo, acompanhando todas as alterações efetuadas. A ferramenta implementa boa parte das funcionalidades do CVS, eliminando alguns inconvenientes. Um dos inconvenientes surge quando o processo de gerenciamento de versões necessita ser interrompido, levando o repositório a um estado inconsistente (LUCENA, 2005). Recursos não disponíveis no CVS também foram incorporados, como por exemplo, versionamento de diretórios, atomicidade nas operações, versionamento de "metadados" (local em que são armazenados dados extras sobre cada arquivo) e tratamento consistente dos dados.

A *Subversion* possui algumas características importantes (LUCENA, 2005):

- Possibilita o controle de versões dos diretórios, das cópias e renomeações. Por exemplo, todas as operações de alteração, remoção e cópia de diretórios são acompanhadas ao longo do tempo e podem ser "desfeitas", assim como ocorre com arquivos individuais. Todo o histórico destas mudanças é acompanhado, em outras palavras, pode-se identificar quando e quem alterou o nome de um diretório.
- Ao contrário de algumas outras ferramentas, as operações são sempre atômicas, ou seja, ou todas as operações são realizadas ou nenhuma delas tem efeito.
- Faz uso de um algoritmo otimizado possibilitando a manipulação eficiente de arquivos texto e binário. Todos os arquivos são armazenados de forma compactada no repositório.

Na *Subversion* o repositório pode ser local ou remoto, sendo que o acesso a esse repositório é feito indiretamente.

Na Tabela 1.1 são apresentados resumidamente os principais aspectos relacionados às ferramentas estudadas nesta seção.

Tabela 1.1 – Comparação das ferramentas estudadas

Ferramentas	Tipos de Arquivos	Características	Plataformas Suportadas
SCCS	Texto	- oferece interface pouco amigável; - não permite o desenvolvimento paralelo; - não permite acesso via rede.	UNIX
RCS	Texto	- oferece interface pouco amigável; - eficiente em sistema de pequeno porte; - desenvolvimento paralelo complicado; - arquivos em um único diretório; - não permite acesso via rede.	UNIX, Windows
CVS	Texto e Binário	- oferece interface amigável; - o repositório pode ser local ou remoto; - permite o desenvolvimento paralelo; - possibilita a criação de uma hierarquia de diretórios dentro de um repositório; - sistema maduro, bem testado e amplamente documentado.	Multi-plataforma
Subversion	Texto e Binário	- oferece interface amigável; - o repositório pode ser local ou remoto; - permite o desenvolvimento paralelo; - atomicidade nas operações; - faz versionamento de diretórios e de *metadados*; - faz tratamento consistente dos dados.	Multi-plataforma

1.3.2. Gerência de Configuração sob a perspectiva de *Frameworks*

O processo de gerenciamento de configuração de *frameworks* deve ser realizado com muita precaução, pois, além de gerenciar as versões do *framework* é necessário também gerenciar as versões das aplicações geradas a partir da sua instanciação, o que tende a aumentar os esforços despendidos e, conseqüentemente, o nível de complexidade (BRAGA, 2003; CAGNIN, 2005).

Na Figura 1.3 é ilustrado o processo de gerenciamento de um *framework* denominado *GREN* (Gestão de Recursos de Negócios) (BRAGA, 2003) e dos sistemas gerados a partir dele. Para facilitar a instanciação do *GREN*, é utilizada a ferramenta de apoio *GREN-Wizard* (BRAGA, 2003).

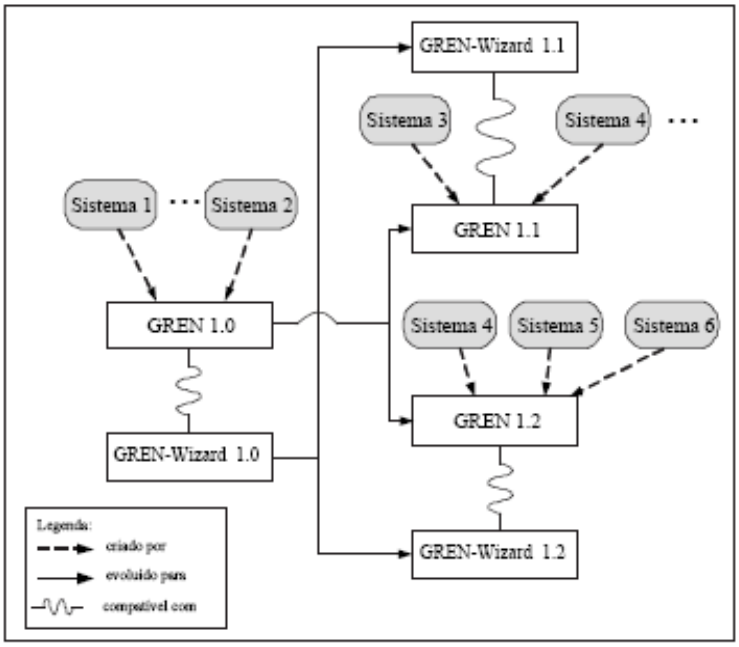


Figura 1.3 – Grafo do controle de versões do *GREN* e dos sistemas gerados (CAGNIN, 2005)

Ressalta-se a importância de manter um registro dos requisitos de cada versão do *framework* com os respectivos sistemas que foram gerados em cada versão, conforme ilustrado na Figura 1.3. Caso contrário, pode haver conflito entre o código fonte do *framework* e o do sistema gerado.

O ciclo de vida de um *framework* é diferente do ciclo de vida de uma aplicação convencional pois, um *framework* não é um componente isolado mas sua existência está diretamente relacionada à existência de outros componentes (SILVA, 2000).

Um *framework*, assim como um software convencional, evolui constantemente. Isto implica em diversas modificações que podem afetar diretamente o projeto do *framework* e das aplicações geradas por ele. Como conseqüência, essas aplicações podem não atender os requisitos pré-estabelecidos e deixar de funcionar corretamente (CAGNIN, 2005). Esta é uma das problemáticas do processo de evolução de *frameworks*. Outro problema é o aumento da complexidade da estrutura do *framework*, o que torna o *framework* mais difícil de ser compreendido e reusado.

Para minimizar esses problemas é preciso definir o histórico de requisitos para armazenar todas as informações relevantes inerentes a evolução de *frameworks* e dos sistemas gerados.

Conforme surgem novas versões, surgem também as dificuldades para se manter a compatibilidade com versões anteriores, uma vez que os *frameworks* atingem um nível de maturidade com o passar do tempo e as aplicações devem acompanhar essa evolução em paralelo. Algumas decisões importantes devem ser tomadas no gerenciamento de configuração de *frameworks* (CAGNIN *et al*., 2004b; CAGNIN, 2005): decidir se as aplicações criadas pelo *framework* devem se adequar para incorporar a evolução do *framework*; determinar em quais circunstâncias a manutenção realizada será incorporada à versão operacional do *framework* e se uma nova funcionalidade deverá ser adicionada ao *framework*. Baseado nisso, o engenheiro responsável deve adotar a melhor solução ou conjunto de soluções aliado a técnicas de SCM e ferramentas de apoio associadas para gerenciar de maneira eficiente, qualquer alteração realizada no decorrer do ciclo de vida do *framework*.

1.3.3. Ferramentas de Apoio ao Gerenciamento de Versões sob a perspectiva de *Frameworks*

Atualmente, existem várias ferramentas de apoio ao SCM, com diferentes funcionalidades e aplicadas em diferentes contextos, como exemplo, as ferramentas apresentadas na seção 1.3.1. Todas elas podem ser usadas para apoiar o gerenciamento de configuração de *frameworks,* porém, nenhuma delas foi desenvolvida especificamente para esse propósito, ou seja, não atende a necessidade de gerenciar tanto as versões dos *frameworks* quanto as aplicações geradas por ele.

Ressalta-se que, até a data de conclusão deste trabalho, não foi encontrada nenhuma ferramenta de controle de versões no contexto de *frameworks*. Existe atualmente, uma ferramenta para controlar as versões geradas por um *framework* específico, descrita a seguir.

- ***GREN – WizardVersionControl***

A ferramenta *GREN-WizardVersionControl* (CAGNIN et. al. 2004a; CAGNIN, 2005) foi desenvolvida para apoiar o gerenciamento de versões das aplicações criadas a partir da instanciação de um determinado *framework*, mais especificamente o *framework GREN* (BRAGA, 2003).

A necessidade da criação da ferramenta *GREN-WizardVersionControl* foi evidenciada com a problemática no gerenciamento das aplicações geradas por *framework*. Essas aplicações, em determinados momentos, precisam passar por algumas adaptações no código fonte e, caso o *framework* seja instanciando novamente para a inclusão de um novo requisito no sistema, todo o código incluído anteriormente no sistema é perdido. No contexto do *framework GREN*, a ferramenta *GREN-WizardVersionControl* evita que isso aconteça (CAGNIN, 2005).

Essa ferramenta oferece suporte para a realização da atividade de gerenciamento de versões no contexto do SCM, contribuindo para a garantia da qualidade do software.

A *GREN-WizardVersionControl* pode ser utilizada constantemente para alterar o código fonte da aplicação gerada pelo *framework GREN*. As alterações são armazenadas em uma base de dados para posterior consulta.

Para garantir que as classes, métodos e outros elementos importantes herdados do *framework* não sejam removidos acidentalmente da aplicação criada a partir da sua instanciação, a ferramenta *GREN-WizardVersionControl* permite remover somente os

elementos que não foram herdados do *framework*. Dessa forma, mantém a integridade dos dados e o correto funcionamento da estrutura da aplicação.

A arquitetura da *GREN-WizardVersionControl* foi baseada na arquitetura do *framework GREN* e da ferramenta de instanciação *GREN-Wizard*. Assim, não houve a necessidade de criar uma camada de integração para permitir a comunicação entre as ferramentas.

Na Figura 1.4 é ilustrada a arquitetura da ferramenta *GREN-Wizard e GREN-WizardVersionControl*, respectivamente.

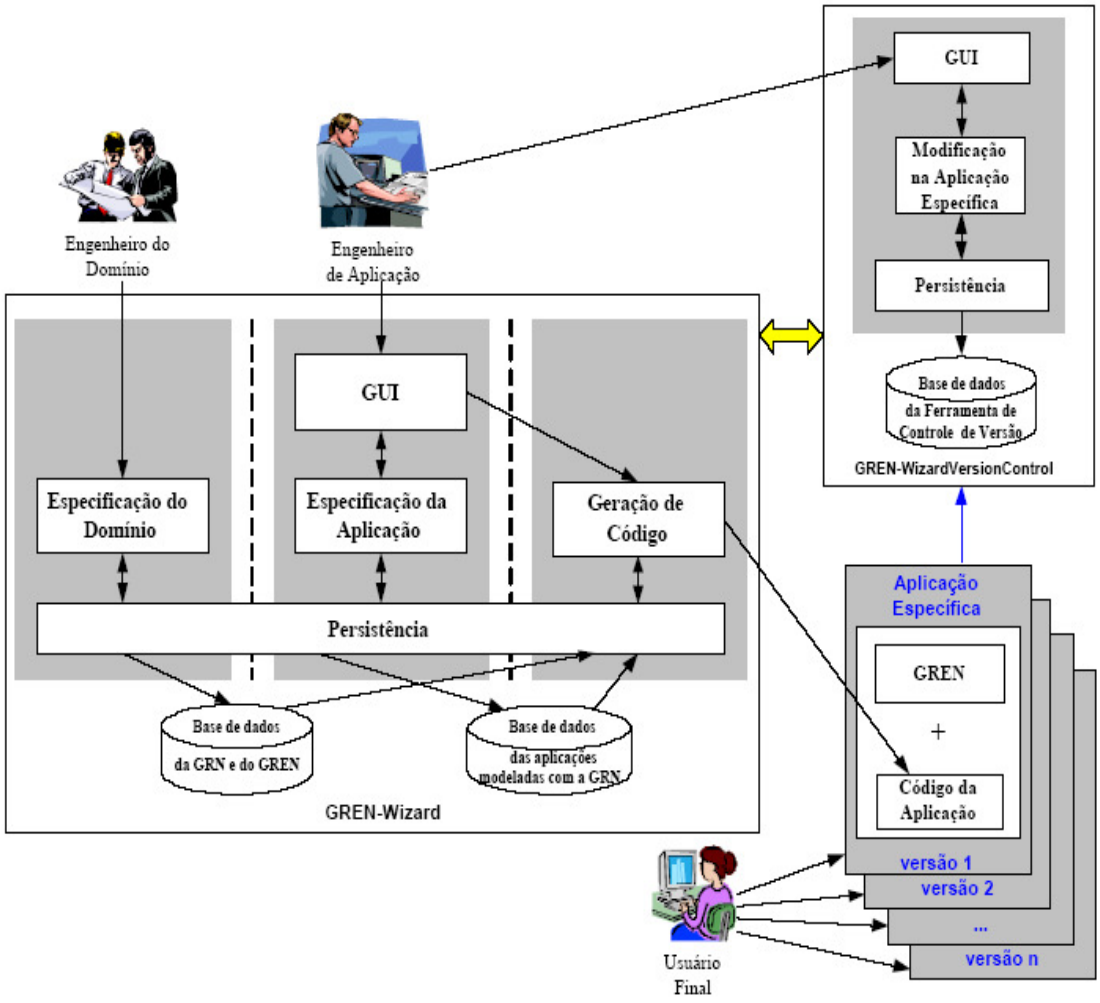


Figura 1.4 – Arquitetura das ferramentas *GREN-Wizard* e *GREN-WizardVersionControl* (CAGNIN, 2005)

Observa-se na Figura 1.4 que, em princípio, o engenheiro de aplicação deve utilizar a *GREN-Wizard* para criar uma primeira versão da aplicação. Se houver a necessidade de modificar manualmente o código fonte da aplicação, a fim de inserir novas funcionalidades não cobertas pelo *framework*, isto deverá ser feito com o auxílio da ferramenta *GREN-WizardVersionControl*, que mantém um histórico de todas as mudanças realizadas em cada versão do sistema. Essas mudanças podem ser incorporadas facilmente às novas versões de cada sistema, criadas pela ferramenta *GREN-Wizard.*

1.4. Considerações Finais

Neste capítulo foram apresentados os conceitos fundamentais necessários para contextualizar o trabalho proposto. Foram coletados os principais trabalhos relacionados à pesquisa sobre *frameworks*, SCM e ferramentas de apoio ao gerenciamento de versões.

Frameworks foram definidos em vários aspectos, relatando o uso dessa tecnologia como um importante mecanismo para facilitar o reúso, aumentar a produtividade, a manutenibilidade e, conseqüentemente, a qualidade do software.

Foram descritos os principais aspectos relacionados ao SCM, enfatizando a importância dessa metodologia no desenvolvimento e evolução de um software. Destaca-se o gerenciamento de versões, que é uma das atividades principais de SCM, bem como o uso de ferramentas computacionais para apoiar esse gerenciamento.

Ressalta-se também que o gerenciamento de configuração sob a perspectiva de *frameworks* é mais complexo, em virtude da necessidade de controlar as versões do *framework* e das aplicações geradas por ele. Para gerenciar as versões das aplicações criadas por um determinado *framework* foi apresentada a ferramenta *GREN-WizardVersionControl*, que pode ser considerada também como uma ferramenta de apoio ao SCM.

Apesar das diversas ferramentas disponíveis para apoiar o SCM, notou-se a ausência de ferramentas específicas de controle de versões no contexto de *frameworks*. A partir desse estudo, observou-se a necessidade de desenvolver uma ferramenta para apoiar esse processo e amenizar essa carência. O Capítulo 2 apresenta a ferramenta proposta neste trabalho.

2. FERRAMENTA DE APOIO AO CONTROLE DE VERSÕES DE *FRAMEWORKS*

2.1. Considerações Iniciais

Neste capítulo apresenta-se a ferramenta denominada TOFRA *(A Framework Version Control Support Tool),* desenvolvida neste trabalho, que apóia o gerenciamento de versões baseado em *frameworks* e busca minimizar a carência de ferramentas computacionais específicas nesse contexto. Na Seção 2.2 apresenta-se um esboço das funcionalidades e da estrutura da ferramenta por meio de diagramas de casos de uso e de um diagrama de classes, respectivamente. Na Seção 2.3 é ilustrada em detalhes a arquitetura da ferramenta. Na Seção 2.4 são apresentadas características detalhadas da implementação da ferramenta. Na Seção 2.5 apresenta-se um estudo de caso para avaliar a ferramenta, a fim de ilustrar suas principais características. Na Seção 2.6 relata-se as dificuldades encontradas durante o desenvolvimento do trabalho e suas limitações. Por fim, na Seção 2.7 apresentam-se as considerações finais sobre este capítulo, com enfoque na ferramenta desenvolvida.

2.2. Modelagem Funcional e Conceitual

A modelagem é uma prática essencialmente importante, sobretudo em sistemas complexos. Modelos ajudam a compreender melhor o sistema em desenvolvimento. Os diagramas de casos de uso apresentados nesta seção foram criados para ilustrar o comportamento geral do sistema sob a perspectiva dos usuários. Ressalta-se que, para facilitar o entendimento do diagrama de casos de uso e melhorar sua visualização, optou-se pela criação dos casos de uso de cada ator (usuário da ferramenta) separadamente.

Para interagir com a ferramenta, foram definidos quatro tipos de usuários: administrador do sistema, proprietário do *framework*, proprietário da aplicação e usuário comum.

O administrador do sistema possui acesso irrestrito a todas as funcionalidades da ferramenta, desde a inserção, remoção e atualização dos dados. Além disso, ele é responsável pelo gerenciamento dos diferentes tipos de usuários e pelas permissões de acesso aos usuários interessados na utilização da ferramenta. O diagrama de casos de uso do administrador do sistema é apresentado na Figura 2.1.

O proprietário do *framework* pode acessar todas as funcionalidades relacionadas aos *frameworks* e às aplicações vinculadas aos *frameworks*, com exceção das operações de remoção e atualização dos dados, que são exclusivas do administrador do sistema. Além disso, esse tipo de usuário pode executar as tarefas comuns, como o processamento de consultas e visualização dos dados, que podem ser executadas por qualquer usuário devidamente cadastrado. Observa-se na Figura 2.2 a interação do proprietário do *framework* com os respectivos casos de usos do qual ele participa.

O proprietário da aplicação tem acesso a todas as funcionalidades relacionadas com as aplicações e ao processamento de consultas e visualização dos dados. Entretanto, esse tipo de usuário não tem a permissão de remover ou alterar qualquer informação. A Figura 2.3 ilustra o diagrama de casos de uso do proprietário da aplicação.

O usuário comum tem uma visão bastante limitada em relação à ferramenta. Seu acesso permite apenas o processamento de consultas e visualização dos dados. Pode-se observar a interação do usuário comum com a ferramenta, por meio do diagrama de casos de uso da Figura 2.4.

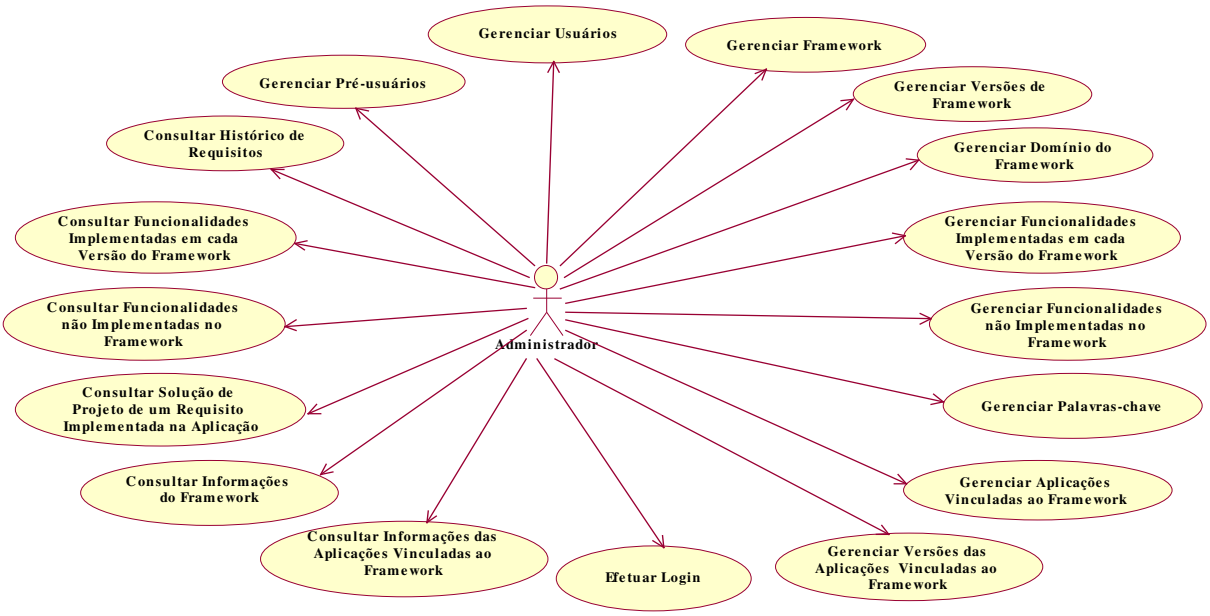


Figura 2.1 – Diagrama de casos de uso do administrador do sistema

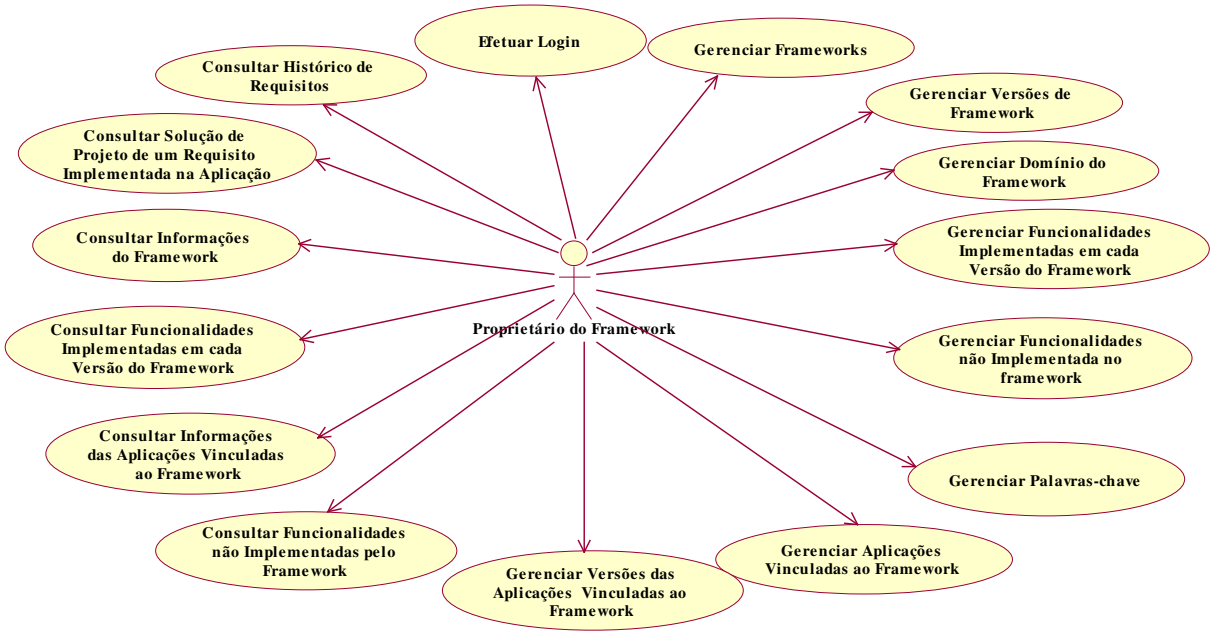


Figura 2.2 – Diagrama de casos de uso do proprietário do *framework*

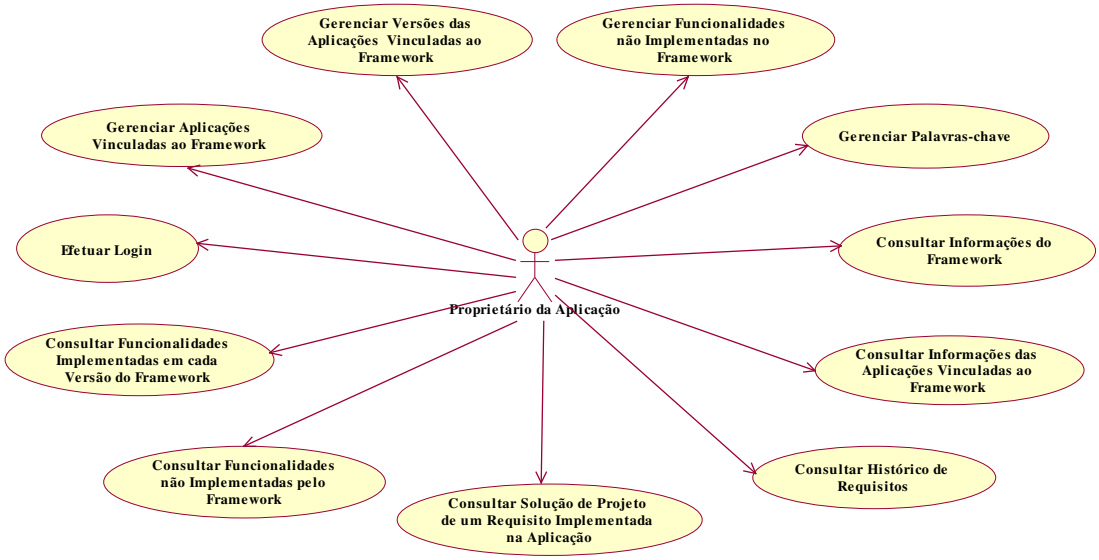


Figura 2.3 – Diagrama de casos de uso do proprietário da aplicação

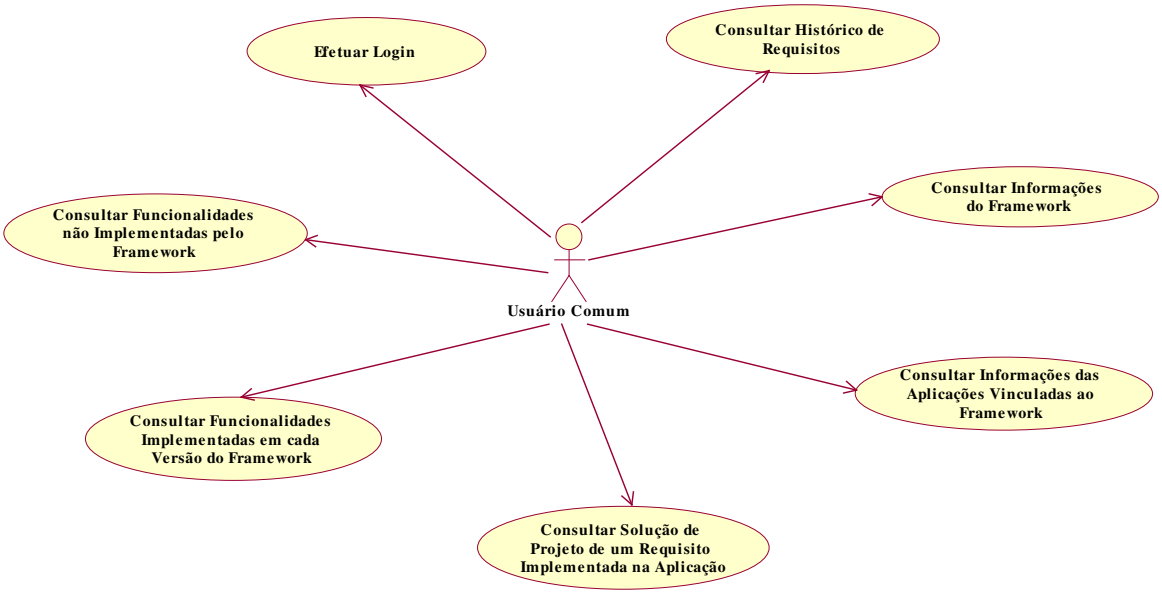


Figura 2.4 – Diagrama de casos de uso do usuário comum

A seguir apresenta-se uma breve descrição de alguns casos de uso. A descrição detalhada encontra-se no (Apêndice B).

O caso de uso “Gerenciar *Frameworks”* tem como objetivo realizar o cadastro dos diferentes *frameworks.*

O caso de uso “Gerenciar Versões de *Frameworks”* tem como objetivo realizar o cadastro das versões dos *frameworks*, que surgem de acordo com o processo de evolução e as necessidades de cada um. Todas as informações pertinentes às versões do *framework*, tais como: o *framework* que gerou a versão, a versão atual do *framework* e a data da criação da versão são tratadas por esta funcionalidade.

O caso de uso “Gerenciar Domínio do *Framework”* tem como objetivo realizar o cadastro do domínio coberto pelo *framework*, ou seja, domínio em que o *framework* foi projetado. Por exemplo, no caso de *frameworks* convencionais, os domínios podem ser, por exemplo, Gestão de Recursos de Negócios e Sistemas de Leilões Virtuais. Já para *frameworks* orientados a aspectos, o domínio pode ser Persistência, Segurança, Distribuição, Concorrência, entre outros (CAMARGO e MASIERO, 2005; CAMARGO, 2006).

O caso de uso “Gerenciar Funcionalidades não Implementadas no *Framework*” tem como objetivo realizar o cadastro dos requisitos (funcionais ou não funcionais) não cobertos pelo *framework*, ou seja, que não foram implementados/atendidos pelo *framework*, e que foram incorporados manualmente em uma determinada aplicação. Todas as informações referentes aos requisitos não cobertos por cada versão do *framework* como, por exemplo, a solução de projeto do requisito implementada na aplicação, a razão da implementação, o nome da aplicação em que o requisito foi incorporado e etc, são tratadas por esta funcionalidade.

O caso de uso “Gerenciar Funcionalidades Implementadas em cada Versão do *Framework”* tem como objetivo realizar o cadastro dos requisitos (funcionais ou não funcionais) cobertos pelo *framework*, ou seja, implementados/atendidos em cada versão do *framework*. Todas as informações pertinentes aos requisitos cobertos por cada versão do *framework* são tratadas por esta funcionalidade.

O caso de uso "Gerenciar Aplicações Vinculadas ao *Framework"* tem como objetivo realizar o cadastro das aplicações vinculadas a cada *framework.*

O caso de uso "Gerenciar Versões das Aplicações Vinculadas ao *Framework"* tem como objetivo realizar o cadastro de determinadas versões de uma aplicação e também vincular essas versões com versões específicas de um ou mais *frameworks*. Essa associação entre versões de aplicações e versões de *frameworks* pode ser vista como um grafo, conforme ilustrado na Figura 2.5. Os nós representam versões de *frameworks* ou de aplicações. Por exemplo, a versão 1 da aplicação 1 (nó V1 APP1) está vinculada à versão 1 do *framework* de persistência orientado a aspectos, à versão 2 do *framework* de autenticação (nó V2 FA) e à versão 3 do *framework* de controle de acesso. Embora essa visualização na forma de um grafo facilite o entendimento de uma determinada configuração das aplicações e dos *frameworks*, ainda não é disponível na ferramenta.

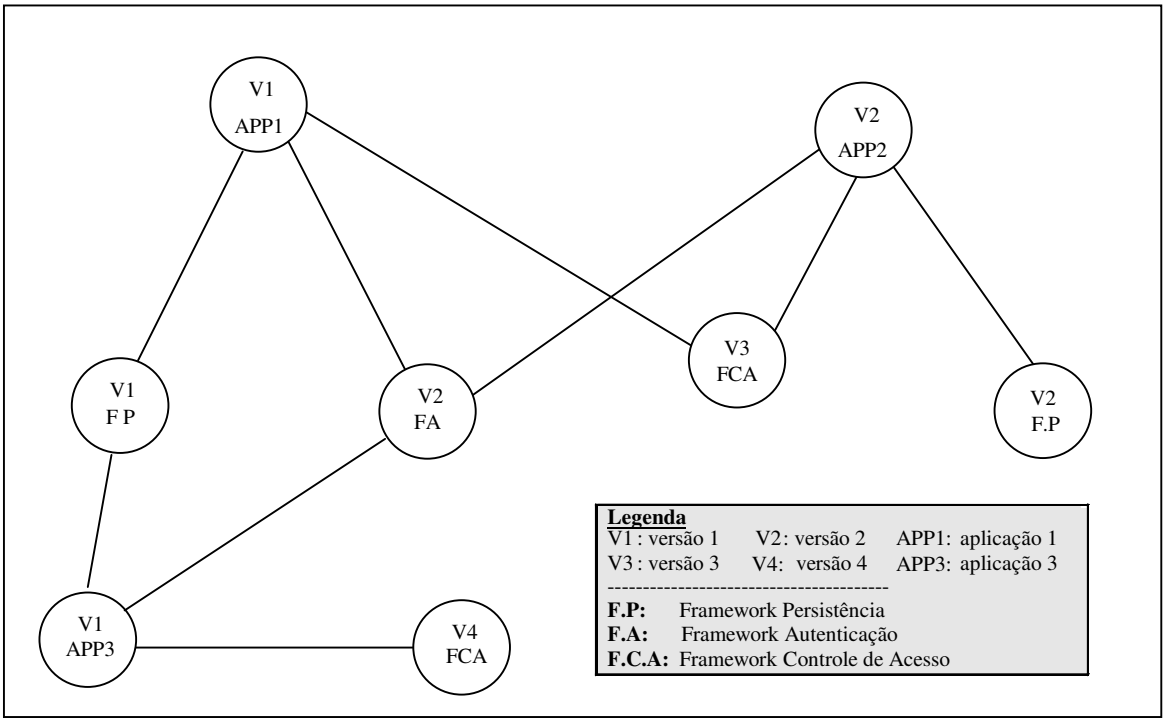


Figura 2.5 – Grafo de associação entre versões de aplicações e versões de FOAs

Há a possibilidade também de uma versão de uma aplicação estar vinculada com versões de *frameworks* convencionais e FOAs. A Figura 2.6 mostra essa associação entre uma versão de uma aplicação com FOO e FOAs. Pode-se observar que, a versão 1 da aplicação 1 (nó V1 APP1) está vinculada à versão 1 do FOO GREN (nó V1 GR), à versão 2 do *framework* de autenticação (nó V2 FA) e à versão 1 do *framework* de persistência (nó V1 FP).

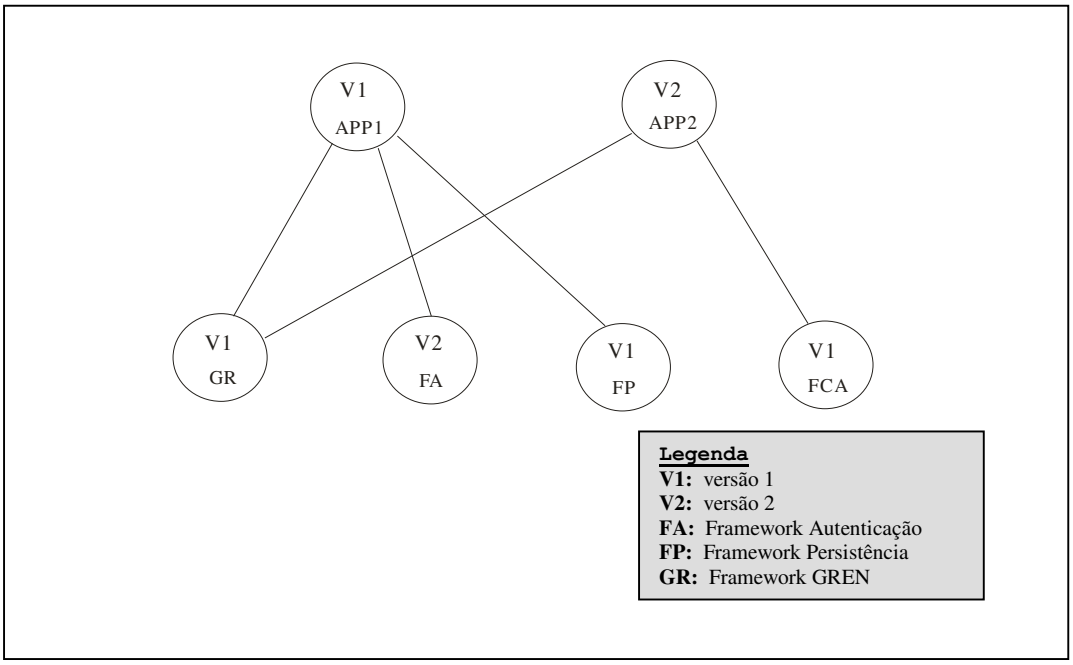


Figura 2.6 – Grafo de associação entre versões de aplicações e versões de FOO e FOAs

O caso de uso "Consultar Funcionalidades não Implementadas no *Framework"* tem como objetivo o processamento de consulta de informações dos requisitos não cobertos pelo *framework* e que foram implementados em uma determinada versão de uma aplicação, como: solução de projeto implementada na aplicação, razão pela qual o requisito não é essencial, tipo de requisito, etc. Nessa consulta o usuário deverá informar a palavra-chave associada ao requisito.

O caso de uso "Consultar Funcionalidades Implementadas em cada Versão do *Framework"* tem como objetivo o processamento de consulta de informações dos requisitos implementados em uma determinada versão do *framework*, como, por exemplo, código de identificação do *framework* que gerou a versão, descrição, etc. Nessa consulta o usuário deverá informar a palavra-chave associada ao requisito.

O caso de uso "Consultar Informações das Aplicações Vinculadas ao *Framework*" tem como objetivo o processamento de consulta de informações das aplicações criadas a partir da instanciação de cada versão do *framework* (no caso de FOO), ou vinculadas a cada versão do *framework* (no caso de FOA).

O caso de uso "Consultar Histórico de Requisitos" tem como objetivo o processamento de consulta no Histórico de Requisitos, que permite visualizar uma lista contendo os requisitos cobertos e não cobertos por cada versão do *framework*, bem como visualizar um requisito para analisar se ele é essencial ao domínio do *framework* e se deve ser incorporado ao mesmo.

A seguir é apresentado na Figura 2.7 o modelo conceitual, representado por meio de diagrama de classes, com o objetivo de fornecer uma visão estática da estrutura do sistema.

A parte superior do diagrama de classes representa o controle de acesso dos usuários, em que cada usuário será identificado por meio de um nome de usuário *(username)* e uma senha *(password).*

Optou-se pela criação do controle de acesso por motivos de segurança, a fim de proteger o conteúdo da base de dados, não permitindo o acesso às informações por parte dos usuários não autorizados.

As classes "*FrameworkVersion"* e "*ApplicationVersion"* são responsáveis pelo controle das versões dos *frameworks* e das versões das aplicações, respectivamente. É a parte em que se vincula as versões das aplicações com as versões de frameworks..

Na parte inferior do diagrama de classes tem-se a representação do Histórico de Requisitos que foi definido para apoiar o PREF (Processo de Evolução de *Frameworks* de Aplicação) (CAGNIN *et al*., 2004; CAGNIN, 2005). No Histórico de Requisitos serão armazenados todos os requisitos (funcionais e não funcionais) cobertos e não cobertos por um determinado *framework.*

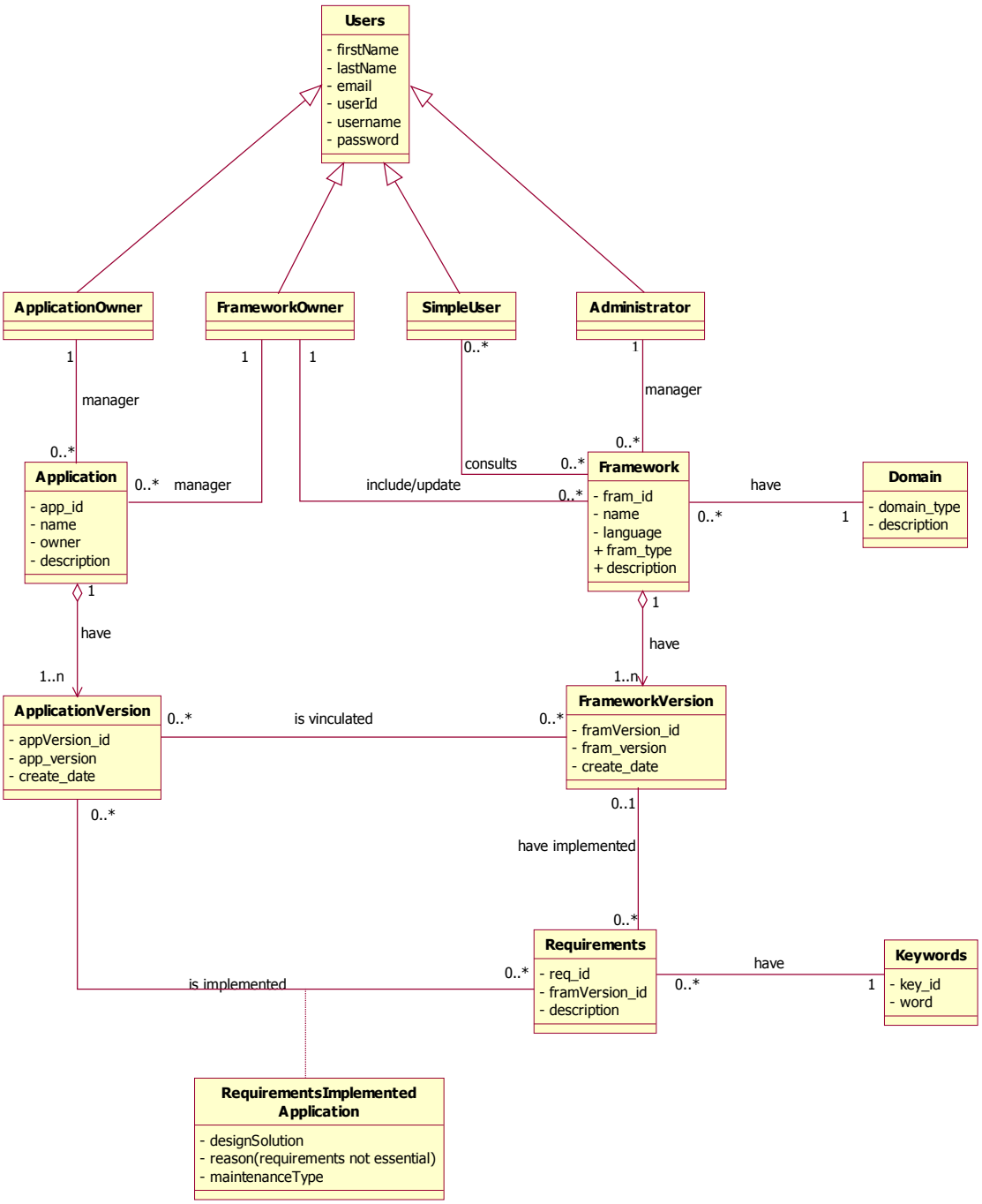


Figura 2.7 – Diagrama de classes

2.3. Arquitetura

A arquitetura da ferramenta desenvolvida é apresentada na Figura 2.8. Um ponto importante a ser ressaltado é que a ferramenta possui arquitetura Web e, assim, pode ser disponibilizada na Internet. O cliente (usuário), por meio do seu navegador (*browser)*, faz a requisição de uma página JSP (*JavaServer Pages)* (SUN MICROSYSTEMS, 2006) ao contêiner JSP (Servidor Web Apache). O servidor web por sua vez faz o processamento da requisição e, se necessário, acessa a base de dados em busca de informações, e por fim retorna a *JavaServer Pages* correspondente. Salienta-se que, o acesso ao banco de dados *Firebird* é feito via JDBC.

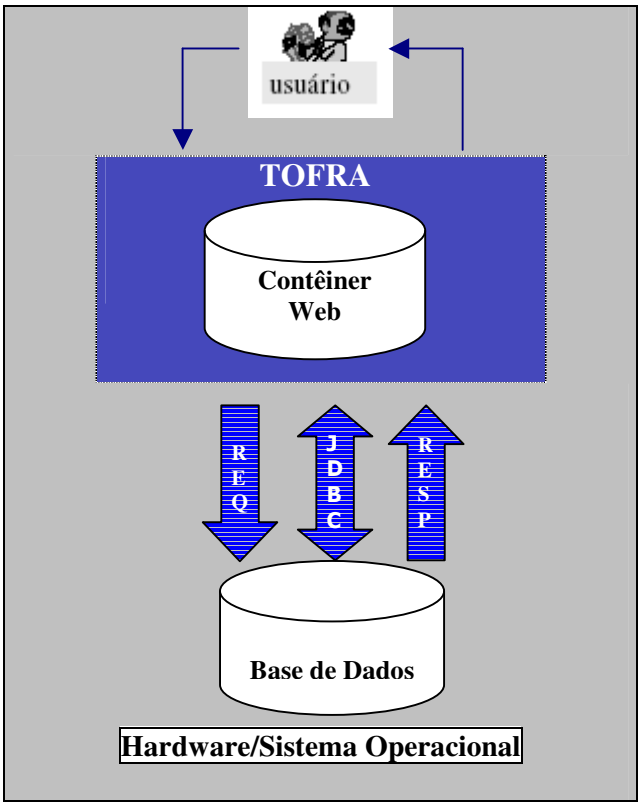


Figura 2.8 – Arquitetura da ferramenta

2.4. Implementação

Após a criação do modelo funcional, conceitual e arquitetônico, iniciou-se a implementação da ferramenta. Para a implementação optou-se pela utilização da linguagem de programação orientada a objetos Java (DEITEL e DEITEL, 2003) e de *JavaServer Pages* (JSP) (TODD, SZOLKOWSKI, 2003) para o desenvolvimento das interfaces. JSP fornece conteúdo dinâmico ao usuário e por ser uma tecnologia Java tem a vantagem da portabilidade, ou seja, pode ser executada em qualquer máquina, independente da plataforma utilizada.

Inicialmente foram definidas todas interfaces da ferramenta e também a base de dados, com o auxílio do Sistema de Gerenciamento de Banco de Dados (SGDB) *Firebird*. Posteriormente implementou-se a autenticação de usuários, por meio de um processo de validação de um nome de usuário *(username)* e uma senha *(password)*. A tela de *login* pode ser visualizada na Figura 2.9. Ao efetuar o *login*, o usuário é direcionado automaticamente para sua página principal, com as funcionalidades no qual ele participa. Cada tipo de usuário tem acesso a um determinado conteúdo, conforme especificado na Seção 2.2. Se o usuário não possui uma conta de acesso no sistema, ele deve inicialmente preencher o formulário de registro *(Register Form)* e aguardar uma autorização do administrador do sistema para utilizar a ferramenta. O administrador do sistema analisa o propósito de cada um e concede ou não a permissão solicitada.

O formulário de registro pode ser acessado a partir do *link Register Form* da Figura 2.9. Nesse formulário os seguintes dados devem ser informados: nome *(firstname),* sobrenome *(lastname)*, email, nome de usuário *(username),* senha *(password),* tipo de usuário *(user type)* e propósito *(purpose).*

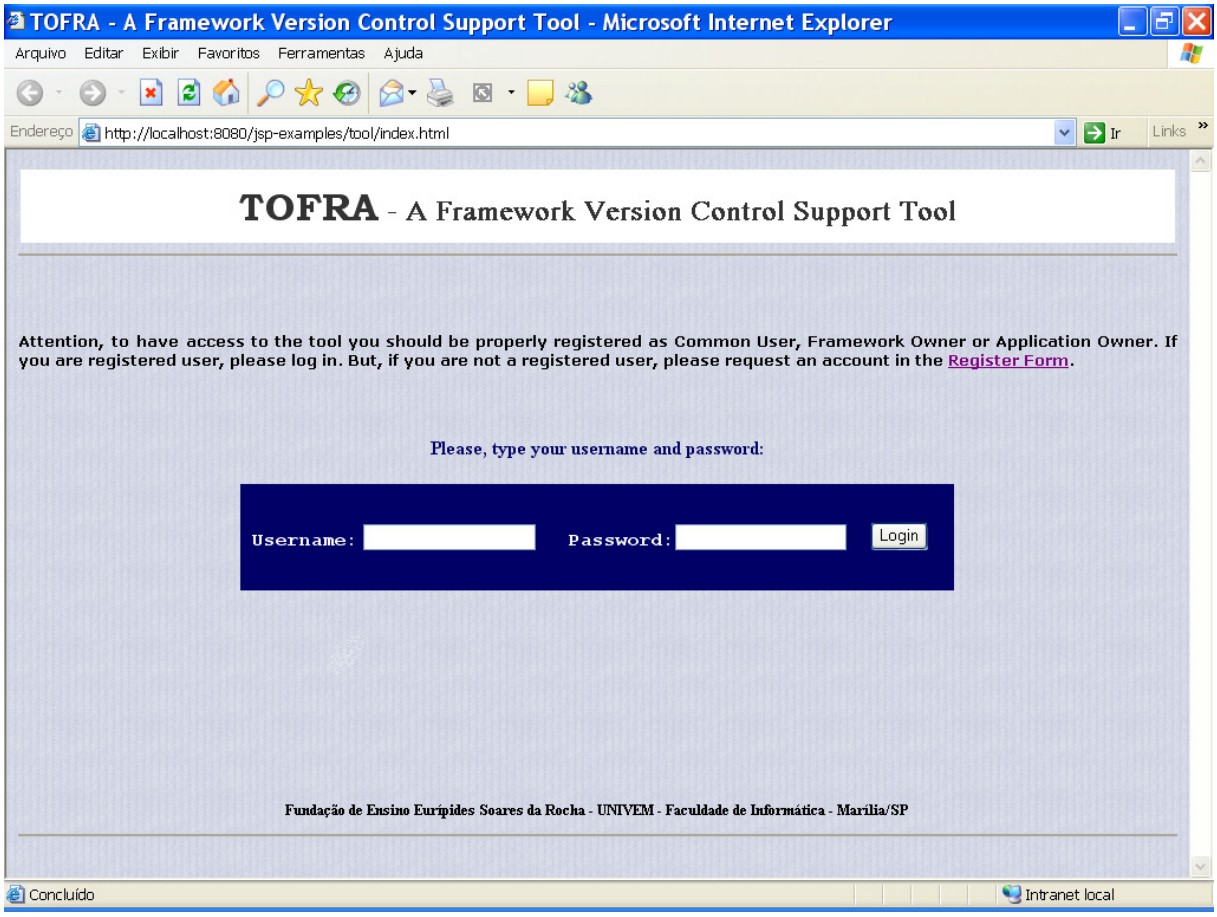


Figura 2.9 – Tela inicial de login

Para utilizar a ferramenta adequadamente há algumas considerações a serem feitas. Observa-se na Figura 2.10, representada pela Notação SADT (*Structured Analysis and Design Technique)* o fluxo de trabalho *(workflow)* que pode ser seguido para a execução de uma determinada atividade na ferramenta. Por exemplo, na atividade "Cadastrar Domínio do *Framework*", um dado de entrada é o domínio do *framework* a ser cadastrado. Um dos caminhos é cadastrar primeiramente o domínio do *framework*, depois o *framework* e assim sucessivamente. Paralelamente também podem ser cadastradas as aplicações, na atividade "Cadastrar Aplicação".

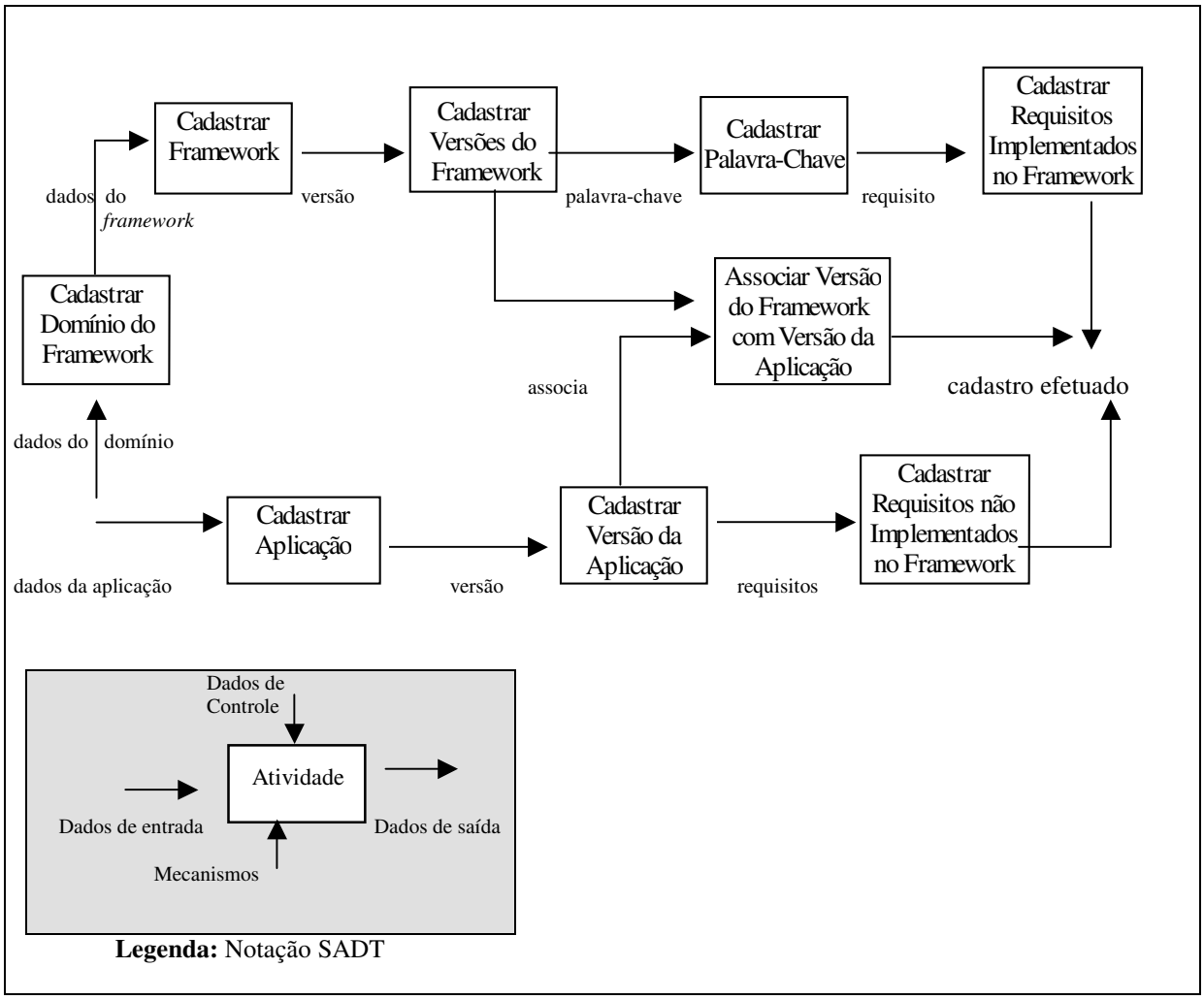


Figura 2.10 – Fluxo de trabalho *(workflow)*

O próximo passo foi implementar os diversos cadastros existentes na ferramenta, como: o cadastro de domínios dos *frameworks,* o cadastro dos *frameworks,* o cadastro de usuários, o cadastro das aplicações vinculadas a cada versão do *framework,* entre outros.

Para efetuar o cadastro de um novo domínio é necessário fornecer apenas o nome/tipo do domínio coberto *pelo framework* e a descrição do domínio.

Na Figura 2.11 é apresentado o formulário de cadastro de *frameworks.* Nessa figura é exemplificada a realização do cadastro do *Framework* de Persistência *(Persistence Framework)* (CAMARGO e MASIERO, 2005; CAMARGO, 2006). Os seguintes dados são necessários: nome do *framework*, proprietário, domínio coberto pelo *framework*, linguagem usada na implementação do *framework*, tipo de *framework (Object Oriented Framework –*

Framework Orientado a Objetos ou Aspect Oriented Framework – Framework Orientado a Aspectos) e a descrição (opcional). Havendo a necessidade de cadastrar o domínio do *framework*, basta acessar o link de cadastro de domínios por meio do botão denominado *Add New Domain*, ilustrado na Figura 2.11.

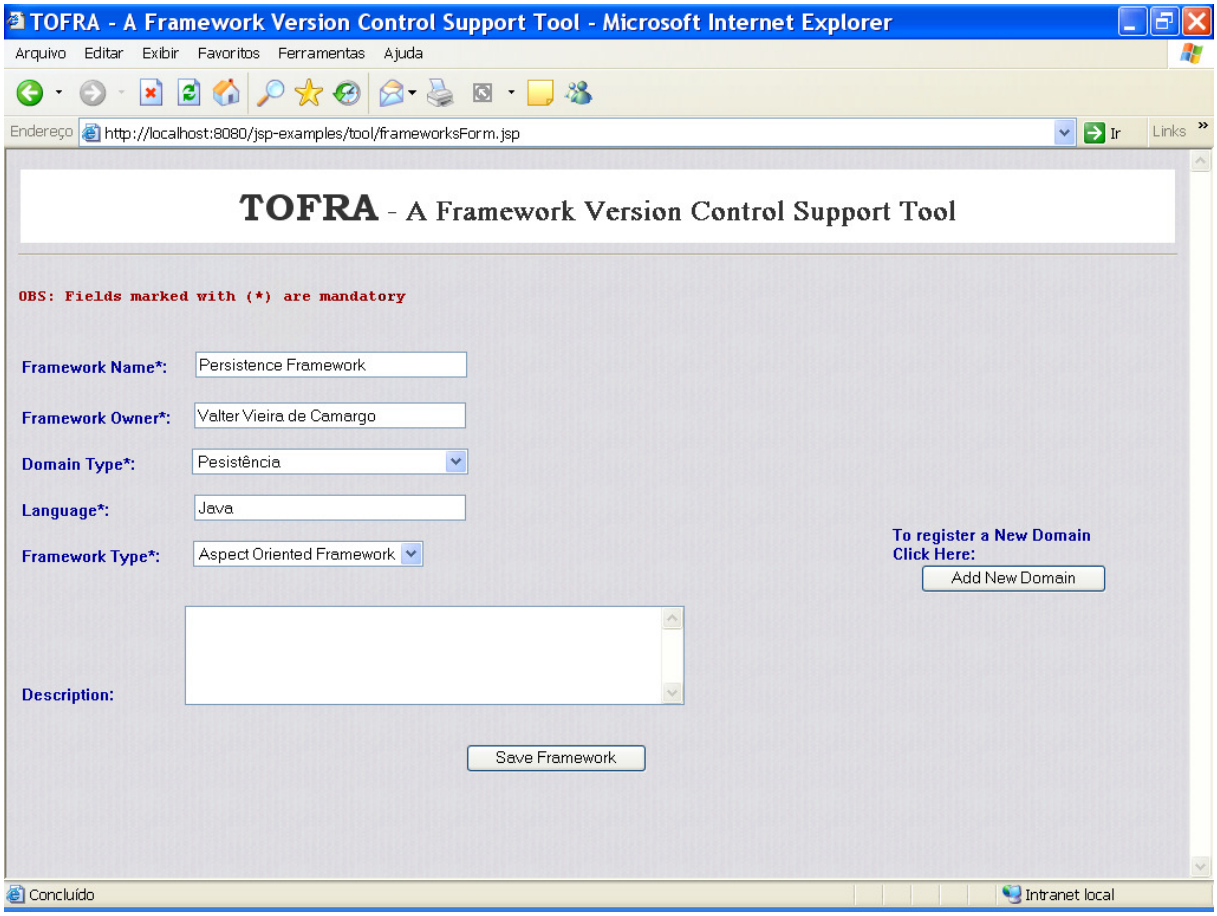


Figura 2.11 – Formulário de cadastro de *frameworks*

Ressalta-se que o controle de versões para sistemas que utilizam FOAs (*Frameworks* Orientados a Aspectos) é ainda mais difícil do que com *frameworks* convencionais, em conseqüência da grande quantidade de *frameworks* que pode ser acoplado a uma única aplicação. Para *frameworks* convencionais geralmente tem-se apenas o vínculo entre um *framework* e uma aplicação. Já no caso de FOAs tem-se uma aplicação com vários

frameworks, o que aumenta ainda mais o nível de complexidade, tanto no contexto de desenvolvimento, quanto no gerenciamento.

Na Figura 2.12 é apresentada a tela de visualização dos *frameworks* cadastrados na base de dados. A partir dessa tela é possível inserir, remover ou alterar os dados de um *framework*, por meio dos botões *Add New Framework* (Adicionar um Novo *Framework*), *Edit Framework* (Editar *Framework* Selecionado) e *Remove Selected* (Remover *Framework* Selecionado). Acessando um desses links, uma nova janela será aberta com o formulário correspondente. Observa-se que, para cada registro há um botão de seleção usado nas operações de alteração e remoção de *frameworks*. Por exemplo, para remover um registro o usuário deve selecionar o *framework* por meio do botão de seleção correspondente e clicar no botão *Remove Selected.*

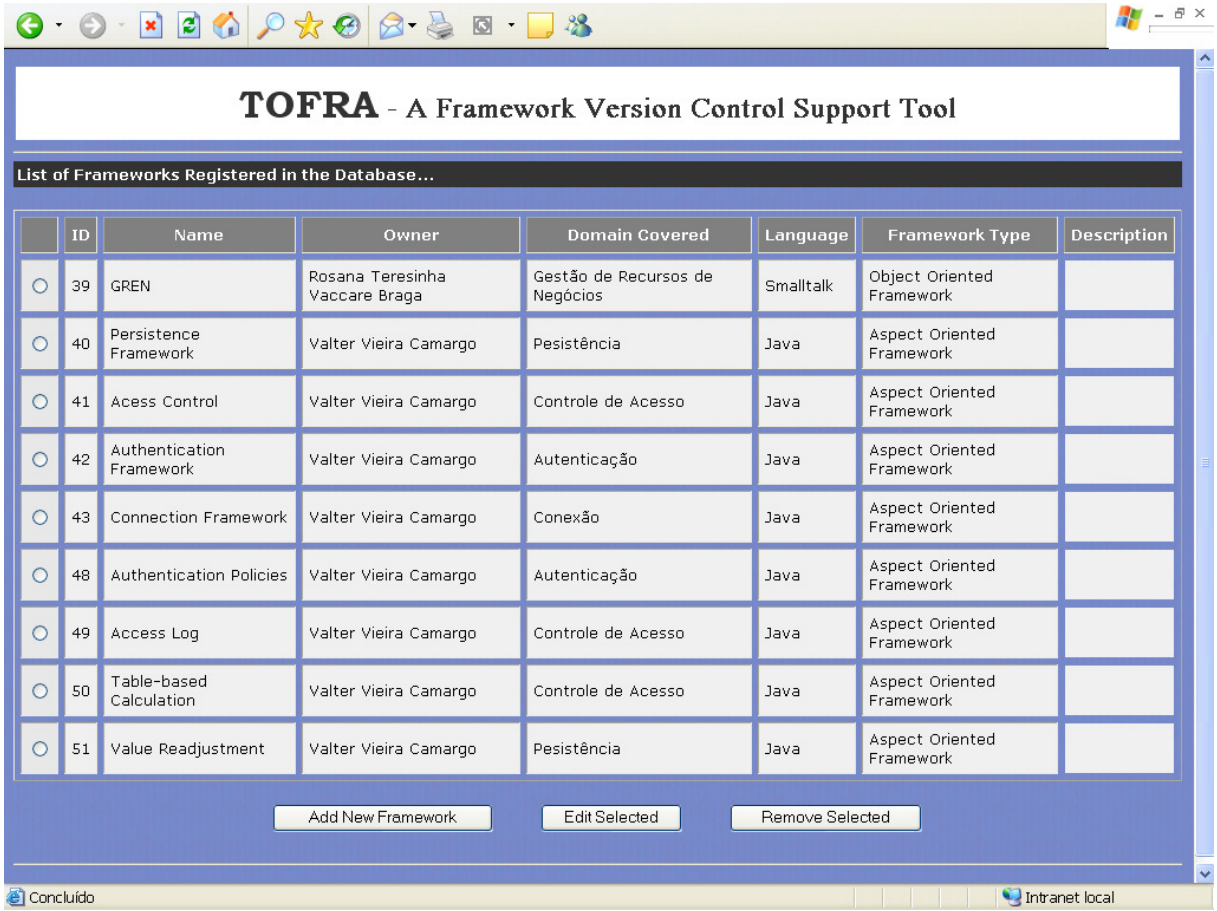


Figura 2.12 – Tela de visualização de *frameworks*

Na Figura 2.13 apresenta-se a tela de visualização de versões de *frameworks*. Para incluir uma nova versão de um *framework* basta clicar no botão *Add New Version*. Em seguida, uma nova janela será aberta com o formulário correspondente. Os seguintes dados são necessários: o código do *framework (framework code),* a versão do *framework (framework version)* e a data da criação da versão *(create date).*

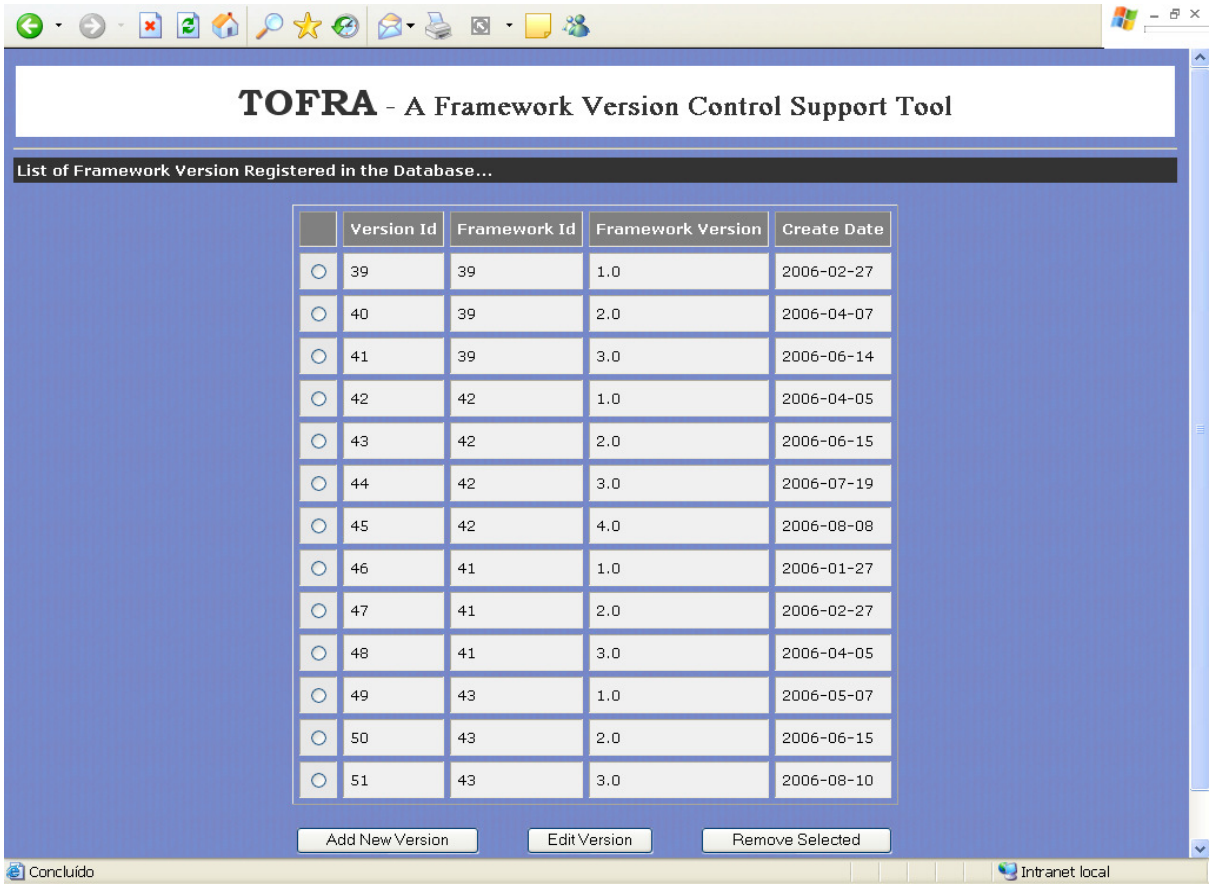


Figura 2.13 – Tela de visualização de versões de *frameworks*

Durante a implementação todos cadastros são validados. Nas operações de inserção dos dados, o sistema não permite que os campos marcados como obrigatórios sejam omitidos ou informações incompletas sejam inseridas e também não permite que os dados sejam modificados por usuários não autorizados, a fim de garantir a consistência e a integridade dos dados.

Na Figura 2.14 é apresentado o formulário de cadastro de usuários. Salienta-se que essa funcionalidade é exclusiva do administrador do sistema.

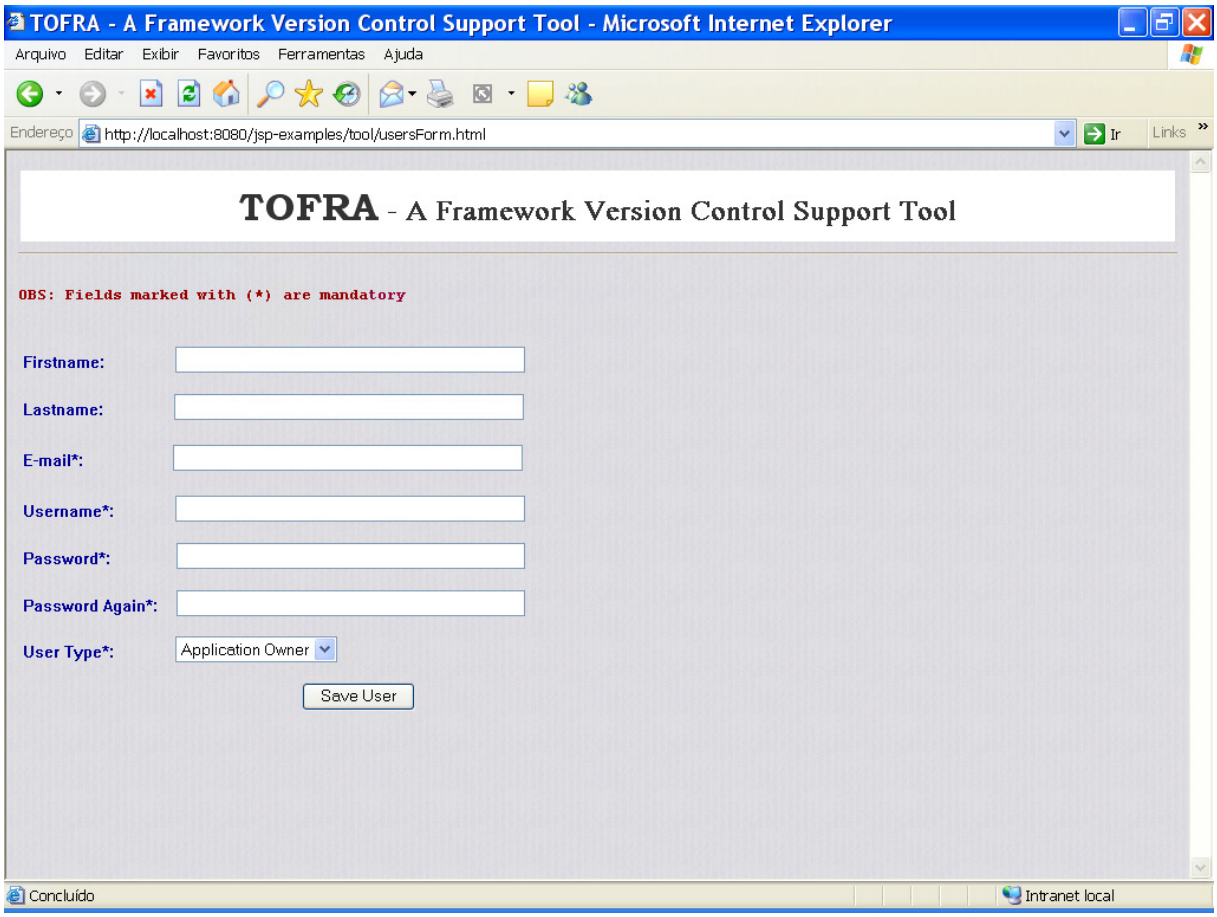


Figura 2.14 – Formulário de cadastro de usuários

No formulário de cadastro de aplicações é necessário fornecer o nome da aplicação, o proprietário da aplicação e a descrição. Para o cadastro de versões de aplicações são necessários os seguintes campos: código/nome da aplicação, versão da aplicação, código/nome do *framework*, versão do *framework* associado a cada versão da aplicação, a data da criação da versão da aplicação e a descrição.

Conforme mencionado na Seção 2.2, para apoiar o PREF (CAGNIN *et. al.*, 2004; CAGNIN, 2005), os requisitos (funcionais e não funcionais) não cobertos pelo *framework*, que foram implementados em alguma versão de uma aplicação são armazenados no Histórico

de Requisitos, a fim de auxiliar na decisão sobre evoluir ou não o *framework*. Além disso, os requisitos cobertos pelo framework, ou seja, implementados em cada versão do *framework* também são armazenados.

O formulário de cadastro de requisitos não cobertos pelo *framework* pode ser visualizado na Figura 2.15.

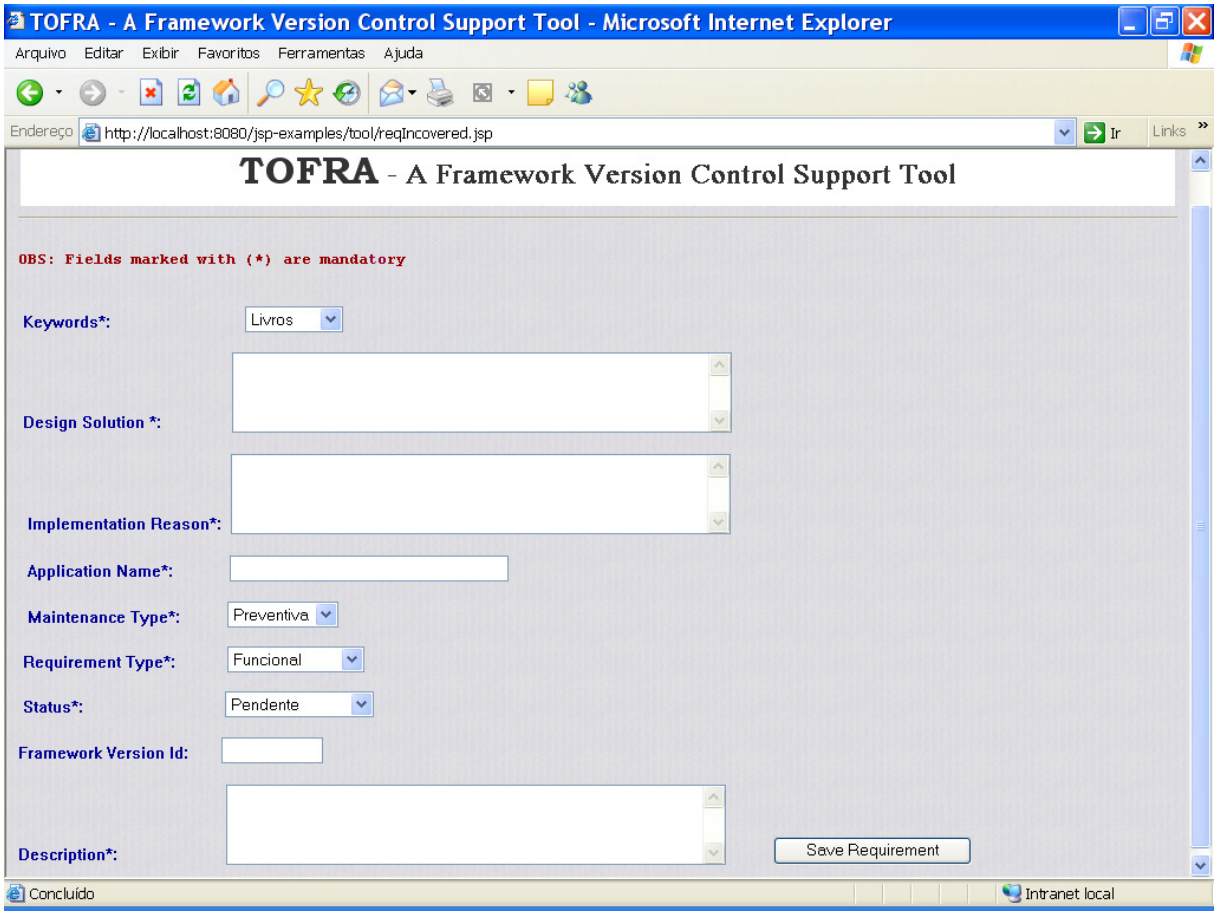


Figura 2.15: Formulário de cadastro de requisitos não cobertos pelo *framework*

Para a realização do cadastro é necessário fornecer a palavra-chave associada ao requisito, a solução de projeto adotado na implementação do requisito na aplicação, a razão da implementação (porque o requisito não é essencial), o nome da aplicação em que o requisito foi implementado, o tipo de manutenção realizada (preventiva, adaptativa, corretiva ou perfectiva), o tipo de requisito (funcional ou não funcional), a situação do requisito em

relação ao framework (estado “pendente” – indica que o requisito ainda não foi incorporado ao framework, estado “sendo atendido” – indica que o requisito está sendo implementado e estado “atendido” – indica que o requisito foi incorporado ao framework), a versão do framework em que o requisito foi incorporado (se for o caso) e a descrição do requisito.

No cadastro de requisitos cobertos pelo *framework* os seguintes dados são necessários: a palavra-chave associada ao requisito, o código de identificação da versão do *framework* e a descrição do requisito.

Na Figura 2.16 apresenta-se a tela de visualização dos requisitos cobertos por um determinado framework. A partir dessa tela é possível inserir, remover ou alterar os dados de um requisito.

Por fim, implementou-se as funcionalidades de consultas a base de dados, como por exemplo: as consultas relacionadas aos requisitos cobertos pelo *framework*, bem como os requisitos não cobertos pelo *framework*, que atendem as exigências do PREF para analisar se o requisito é essencial ou não ao domínio do *framework*.

Para analisar se um requisito é essencial ao domínio do *framework* é necessário seguir algumas diretrizes exigidas pelo PREF:

1 – Deve-se utilizar o conhecimento no domínio;

2 – Deve-se construir uma lista com as prováveis situações em que um requisito pode ocorrer novamente. Se existir pelo menos duas situações em que o requisito pode ocorrer novamente é bem provável que ele é essencial ao domínio do *framework* e deve ser incorporado;

3 – Deve-se consultar o Histórico de Requisitos para verificar a existência de mais duas ocorrências do requisito em questão, cuja situação esteja no estado “pendente”. Se esse for o caso, o requisito provavelmente é essencial ao domínio do *framework* e deve ser incorporado.

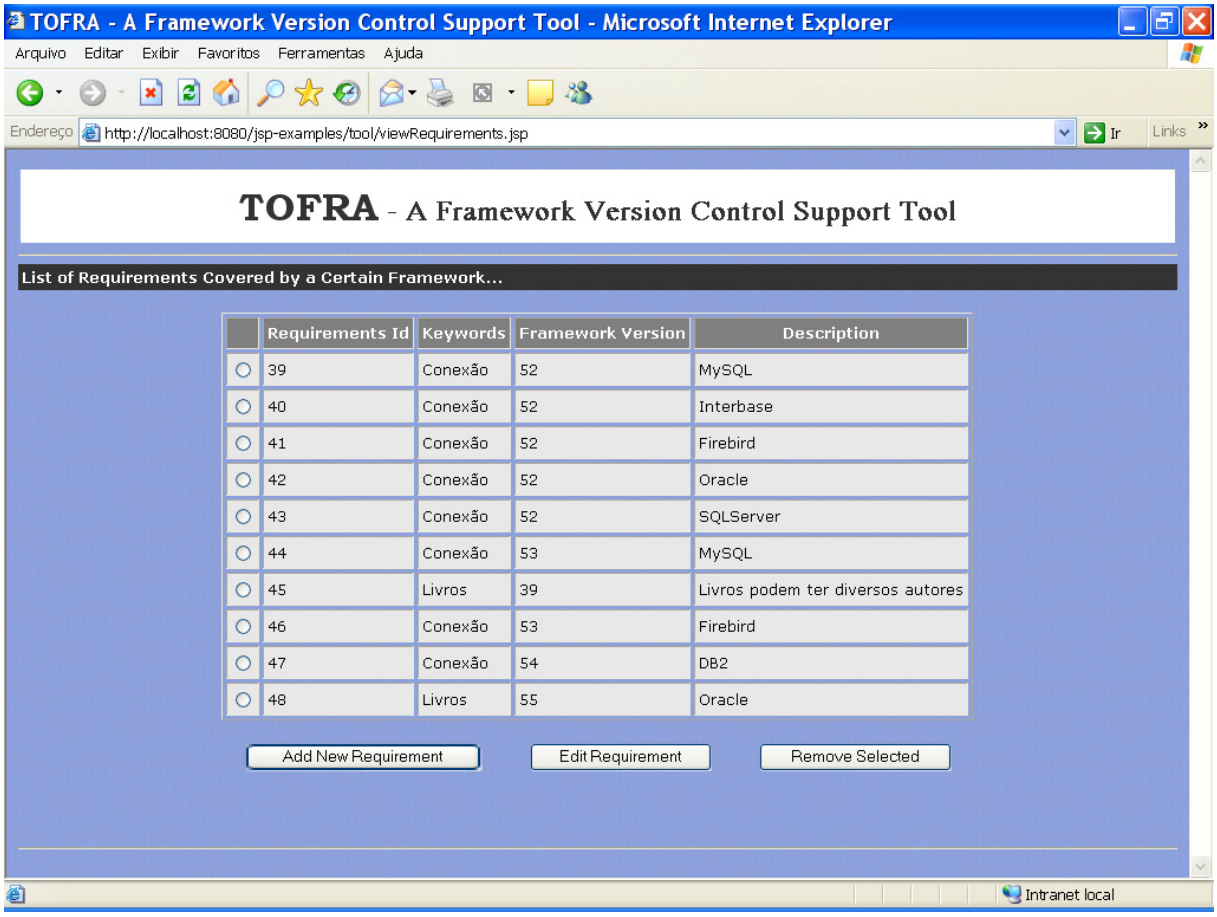


Figura 2.16 – Tela de visualização de requisitos cobertos pelo *framework*

É importante destacar que a implementação das funcionalidades da ferramenta desenvolvida foi realizada de forma mais genérica, de modo que os códigos possam ser reutilizados facilmente por outros desenvolvedores.

2.5. Estudo de Caso

O estudo de caso apresentado nesta seção tem por finalidade testar a ferramenta TOFRA.

Na Tabela 2.1 apresentam-se algumas aplicações desenvolvidas em uma Tese de Doutorado e os FOAs que foram utilizados em cada aplicação (CAMARGO, 2006). Esses

frameworks e algumas dessas aplicações foram utilizados para testar a ferramenta. Além disso, criou-se uma aplicação "hipotética" denominada Biblioteca, apenas para exemplificar a associação entre um *framework* convencional, no caso o GREN com uma única aplicação.

A Tabela 2.1 apresenta as aplicações e os *frameworks* utilizados, mas não apresenta as versões de cada aplicação e nem as versões de cada *framework*. Sendo assim, uma breve descrição é mostrada nos parágrafos a seguir.

Tabela 2.1 – Aplicações vinculadas com FOAs (CAMARGO, 2006)

Aplicações / *Frameworks*	Oficina de Aparelhos Eletrônicos	Folha de Pagamento	Loja de Venda de CDs	Gerenciamento de Contas Bancárias	Caixa Eletrônico	Hotel
Persistence	✠	✠	✠	✠		✠
Caching – all tuples	✠		✠			✠
Caching – object loader	✠		✠			✠
Connection	✠	✠	✠	✠		✠
Pooling	✠		✠			✠
Access control	✠			✠		✠
Authentication	✠			✠		✠
Authentication policies	✠			✠		✠
Access log	✠			✠		✠
Table-based calculation		✠			✠	✠
Value readjustment		✠	✠			✠
Tracing	✠	✠	✠	✠		✠
Policy enforcement	✠	✠	✠	✠		

Na versão 1.0 da aplicação Biblioteca foi utilizada somente a versão 1.0 do *Framework* GREN.

Na versão 1.0 da aplicação "Oficina de Aparelhos Eletrônicos" foram utilizadas as seguintes versões de *frameworks*: *Persistence 1.0, Caching-all tuples 2.0, Caching-object loader 3.0, Connection 3.0, Pooling 4.0, Access Control 5.0, Authentication 6.0, Access Log 5.0, Tracing 1.0 e Policy Enforcement 4.0.* Entretanto, na versão 2.0 da mesma aplicação utilizou-se o *Authentication 5.0, Persistence 4.0, Policy Enforcement 4.0, Tracing 1.0 e*

Access Control 5.0. Na Figura 2.17 podem ser visualizadas as versões de frameworks utilizadas na aplicação "Oficina de Aparelhos Eletrônicos" versão 1.0 e 2.0 respectivamente.

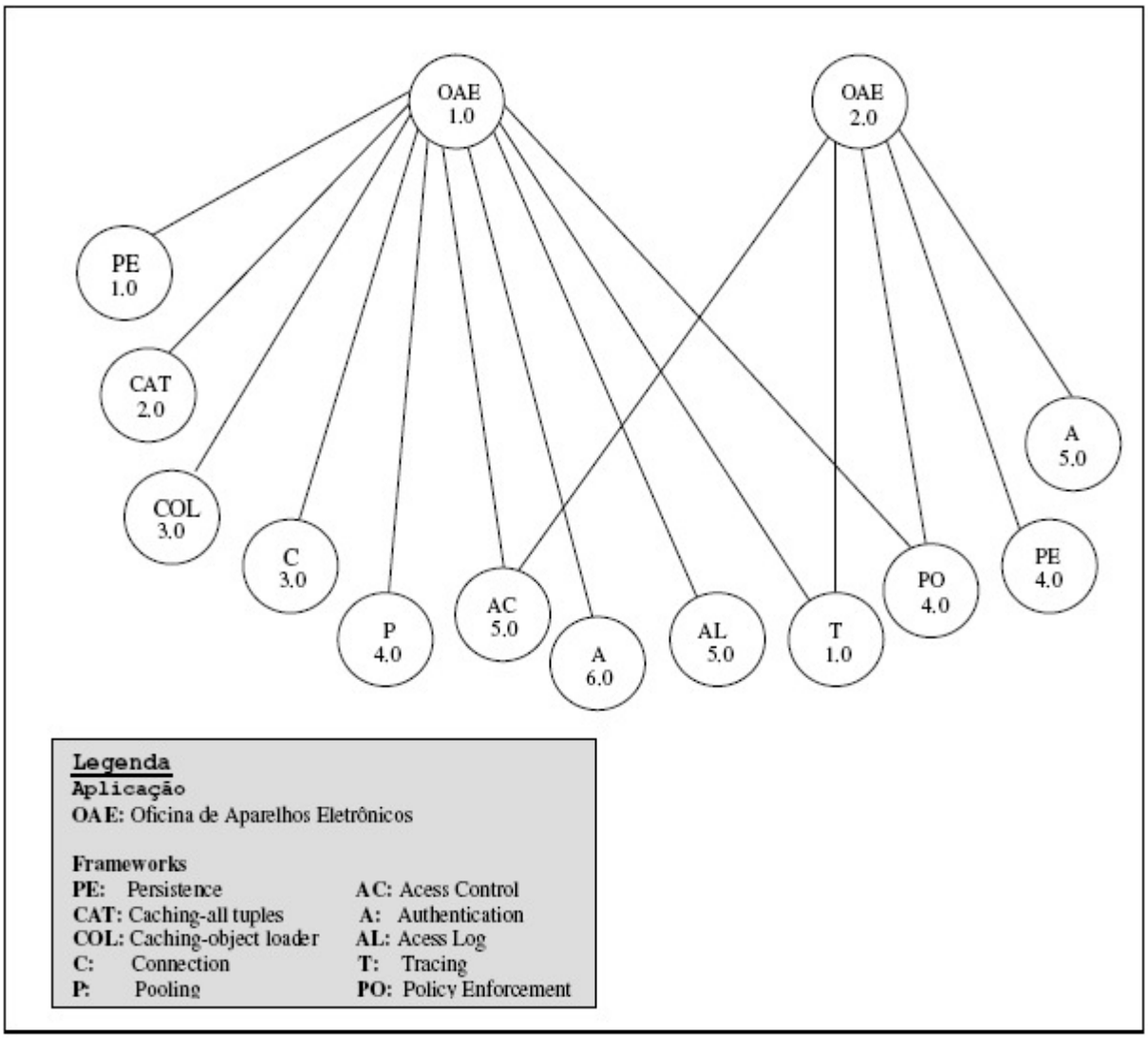


Figura 2.17 – Grafo de associação entre a aplicação oficina de aparelhos eletrônicos e as versões de frameworks utilizadas no seu desenvolvimento

Na versão 1.0 da aplicação "Folha de Pagamento" foram utilizadas as seguintes versões de *frameworks*: *Persistence 1.0, Connection 2.0, Table-based Calculation 3.0, Value Readjustment 3.0, Tracing 4.0 e Policy Enforcement 5.0.* Essa associação pode ser visualizada por meio do grafo da Figura 2.18.

Na Figura 2.19 é apresentada a aplicação "Gerenciamento de Contas Bancarias" e as respectivas versões de *frameworks* utilizadas no seu desenvolvimento.

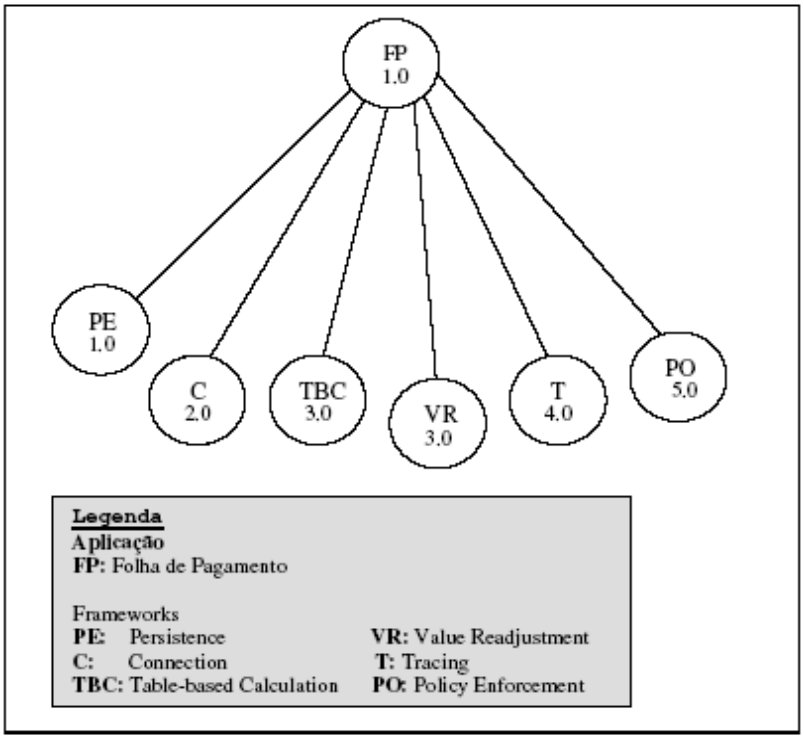


Figura 2.18 – Grafo de associação entre a aplicação folha de pagamento e as versões de frameworks utilizadas no seu desenvolvimento

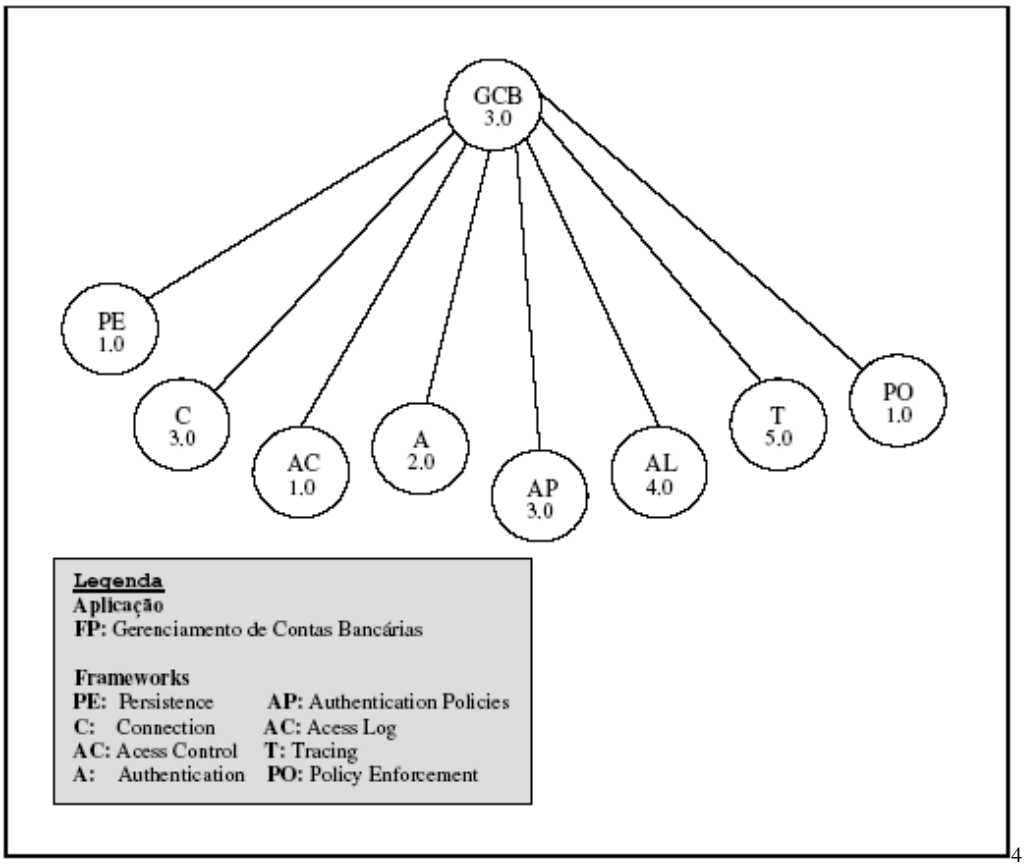
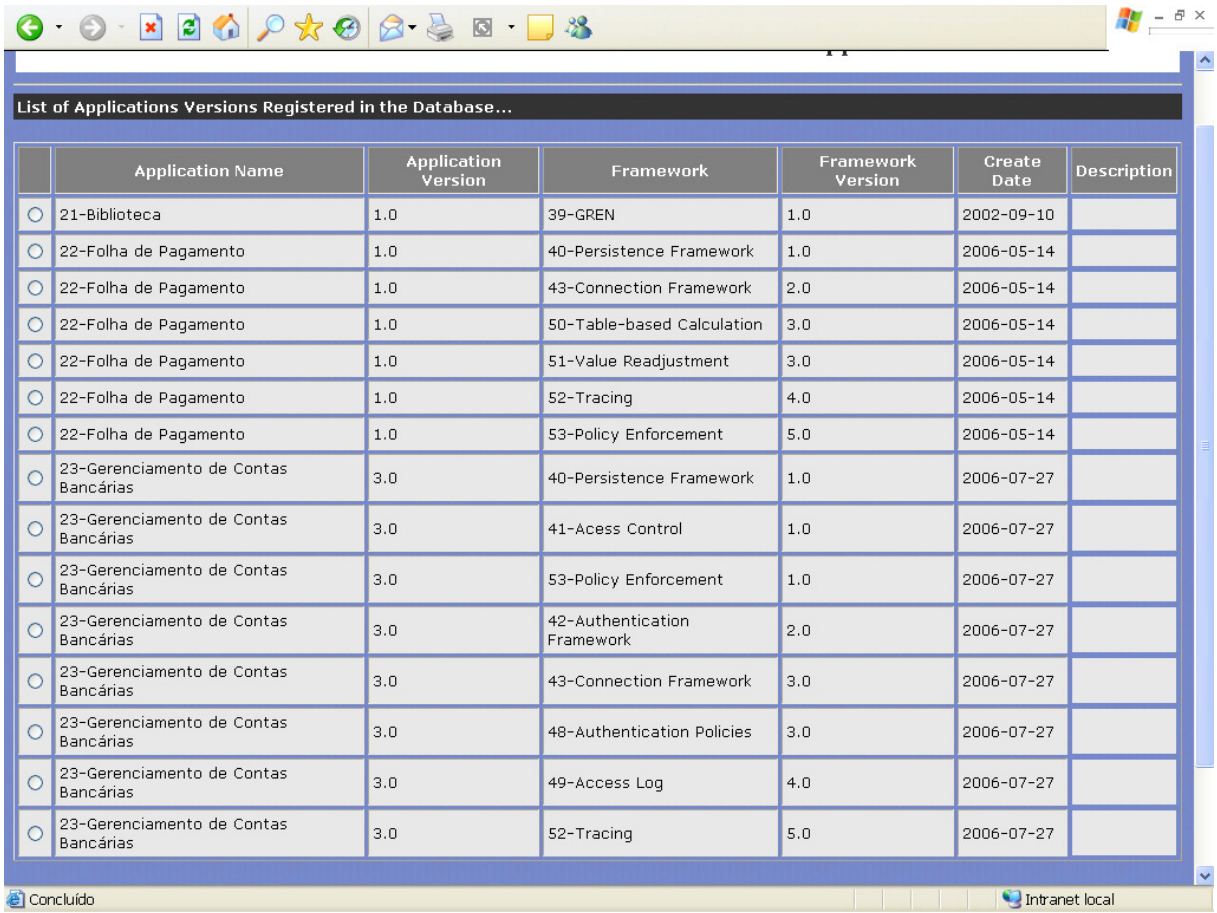


Figura 2.19 – Grafo de associação entre a aplicação gerenciamento de contas bancárias e as versões de frameworks utilizadas no seu desenvolvimento

Finalmente na Figura 2.20 é possível visualizar os *frameworks* que foram utilizados no desenvolvimento de cada aplicação.

Por exemplo, a aplicação "Folha de Pagamento" cuja versão é 1.0 possui vínculo com os seguintes Frameworks: P*ersistence 1.0, Connection 2.0, Table-based Calculation 3.0, Value Readjustment 3.0, Tracing 4.0 e Policy Enforcement 5.0.*

A aplicação "Gerenciamento de Contas Bancárias" versão 3.0 está vinculada com os seguintes frameworks: *Persistence* 1.0, *Connection* 3.0, A*ccess Control 1.0, Authentication* 2.0, *Authentication Policies* 3.0, *Access Log* 4.0, *Tracing* 5.0 e *Policy Enforcement* 1.0. Diferentemente da aplicação Biblioteca que possui vínculo apenas com o *framework* GREN.



List of Applications Versions Registered in the Database...

	Application Name	Application Version	Framework	Framework Version	Create Date	Description
○	21-Biblioteca	1.0	39-GREN	1.0	2002-09-10	
○	22-Folha de Pagamento	1.0	40-Persistence Framework	1.0	2006-05-14	
○	22-Folha de Pagamento	1.0	43-Connection Framework	2.0	2006-05-14	
○	22-Folha de Pagamento	1.0	50-Table-based Calculation	3.0	2006-05-14	
○	22-Folha de Pagamento	1.0	51-Value Readjustment	3.0	2006-05-14	
○	22-Folha de Pagamento	1.0	52-Tracing	4.0	2006-05-14	
○	22-Folha de Pagamento	1.0	53-Policy Enforcement	5.0	2006-05-14	
○	23-Gerenciamento de Contas Bancárias	3.0	40-Persistence Framework	1.0	2006-07-27	
○	23-Gerenciamento de Contas Bancárias	3.0	41-Acess Control	1.0	2006-07-27	
○	23-Gerenciamento de Contas Bancárias	3.0	53-Policy Enforcement	1.0	2006-07-27	
○	23-Gerenciamento de Contas Bancárias	3.0	42-Authentication Framework	2.0	2006-07-27	
○	23-Gerenciamento de Contas Bancárias	3.0	43-Connection Framework	3.0	2006-07-27	
○	23-Gerenciamento de Contas Bancárias	3.0	48-Authentication Policies	3.0	2006-07-27	
○	23-Gerenciamento de Contas Bancárias	3.0	49-Access Log	4.0	2006-07-27	
○	23-Gerenciamento de Contas Bancárias	3.0	52-Tracing	5.0	2006-07-27	

Concluído

Intranet local

Figura 2.20 – Tela de visualização de versões de aplicações

Nota-se que, à medida que o número de versões de *frameworks* e o número versões de aplicações aumentam, os esforços despendidos no gerenciamento destas versões também aumentam consideravelmente. Por exemplo, se cada *framework* tiver três versões e cada aplicação cinco versões, o controle torna-se realmente muito complexo.

Com o apoio da ferramenta TOFRA foi possível manter de forma sistemática o controle das versões das aplicações vinculadas a cada versão de um *framework*, ressaltando ainda mais a importância de se utilizar uma ferramenta no controle de versões sob essa perspectiva.

2.6. Dificuldades e Limitações

Algumas limitações dessa pesquisa são decorrentes principalmente, do fato de que, poucos estudos de casos foram conduzidos, em virtude do pouco tempo disponível. Salienta-se a necessidade da realização de novos estudos de casos.

A ferramenta também foi utilizada somente no controle de versões de aplicações baseadas em *frameworks.* Dessa forma, não foi possível verificar a sua aplicabilidade em outras técnicas de reúso de software.

2.7. Considerações Finais

A problemática no controle de versões de aplicações baseadas em *framework* pôde ser observada no estudo de caso realizado na Seção 2.5. Nesse experimento, ficou mais evidente a necessidade de se utilizar ferramentas que apóie todo esse gerenciamento.

A ferramenta TOFRA é útil não somente no controle de versões de FOOs mas, principalmente em FOAs, em que as dificuldades são ainda maiores já que uma versão de uma aplicação pode estar associada a várias versões de *frameworks* diferentes.

No próximo capítulo são apresentadas as conclusões obtidas com o desenvolvimento deste trabalho, ressaltando os resultados obtidos e as contribuições, além de sugestões de alguns trabalhos decorrentes desta pesquisa que podem ser realizados futuramente.

3. Conclusões

3.1. Resultados Obtidos e Contribuições

O nível de complexidade no controle de versões de aplicações baseadas em *frameworks* é maior, sobretudo em *frameworks* orientados a aspectos, conforme abordado nessa pesquisa.

A TOFRA veio suprir a carência de ferramentas computacionais de apoio ao controle de versões de aplicações baseadas em *frameworks,* uma vez que, não foi encontrada na literatura, nenhuma ferramenta de controle de versões sob essa perspectiva.

Uma característica importante é que a ferramenta permite controlar as versões de *frameworks* orientados a aspectos, em que mais de um *framework* pode ser vinculado a uma determinada aplicação, o que não acontece com os *frameworks* convencionais.

Os resultados obtidos com o estudo de caso realizado na Seção 2.5 mostram que, com o apoio da ferramenta desenvolvida, os problemas investigados durante a realização desta pesquisa são minimizados. A ferramenta oferece funcionalidades que permitem armazenar de forma organizada todas as informações pertinentes aos *frameworks* e as aplicações vinculadas, como, por exemplo, uma lista dos requisitos cobertos e não cobertos por um determinado *framework*. Tais informações são importantes na decisão de evoluir ou não um *framework*, com o auxílio do PREF (CAGNIN, 2005).

Com os experimentos foi possível ainda avaliar o desempenho e a usabilidade da ferramenta tanto em *frameworks* orientados a objetos, como em *frameworks* orientados a aspectos reais (*Framework* de Autenticação, *Framework* de Controle de Acesso, *Framework* de Persistência, entre outros*)* (CAMARGO e MASIERO, 2005; CAMARGO, 2006).

A principal contribuição pertinente a esta pesquisa refere-se não apenas ao fornecimento de uma ferramenta computacional, mas de uma forma sistematizada de se realizar o controle de versões de aplicações baseadas em *frameworks*. A ferramenta auxilia no controle das mudanças realizadas e na redução dos problemas identificados no processo de gerência de configuração, contribuindo para a garantia de qualidade do software.

Outra característica a ser ressaltada refere-se a sua arquitetura, que permite ser disponibilizada via Web, tornando-se facilmente acessível às pessoas geograficamente dispersas.

3.2. Trabalhos Futuros

Como perspectiva para trabalhos futuros, seguem algumas sugestões:

- Reestruturar a ferramenta desenvolvida utilizando um modelo de padrão de projeto, como por exemplo, o MVC *(Model View Controller).*
- Implementar o grafo de associação das versões dos *frameworks* com as versões das aplicações. O processo de associação na forma de um grafo facilita a visualização e o entendimento.
- Conduzir novos estudos de casos com outros *frameworks* de aplicação e *frameworks* orientados a aspectos.
- Investigar outras técnicas de reutilização de software disponíveis atualmente.
- Verificar a aplicabilidade da ferramenta no controle de versões de geradores de aplicações.
- Verificar a aplicabilidade da ferramenta em sistemas desenvolvidos com paradigma de componentes.

- Estender/testar a ferramenta no sentido de atender todas as necessidades dos *frameworks* orientados a aspectos, uma vez que esses *frameworks* também podem ser acoplados (vinculados) à outros *frameworks*, e não somente a aplicações.
- Automatizar as inclusões, por exemplo, as inclusões de aplicações. Uma alternativa interessante seria o desenvolvimento de uma ferramenta de instanciação para auxiliar no processo de automatização.

REFERÊNCIAS BIBLIOGRÁFICAS

APACHE SOFTWARE FOUNDATION. **Apache Tomcat.** Disponível em: <http://www.apache.org/>. Acesso em: 11 jun. 2006.

BAR, M.; FOGEL, K. **Open Source Development with CVS.** 3rd ed. Paraglyph Press, Scottsdale, Arizona, 2003. 368p.

BOLINGER, D.; BRONSON, T. **Applying RCS and SCCS – From Source Control to Project Control**. Sebastopol, v.1, n.1, 520p., September, 1995.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML: Guia do Usuário**. 1 ed. Rio de Janeiro: Campus, 2000. 472p.

BOSCH, J.; MOLIN, P.; MATTSSON, M.; BENGTSSON, P. **Object-Oriented *Frameworks* – Problems & Experiences**. Ronneby: 1997. 20f. Research Report 9/97 – Department of Computer Science and Business Administration, University of Karlskrona/Ronneby, 1997.

BRAGA, R. T. V. **Um Processo para Construção e Instanciação de *Frameworks* baseados em uma Linguagem de Padrões para um Domínio Específico**. 2003. Tese (Doutorado em Ciência da Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2003.

BRAGA, R. T. V.; MASIERO, P. C. Finding Frameworks Hot Spots in Pattern Languages. **Journal of Object Technology**, v.3, n.1, p.123-142, January-February. 2004.

BROECKE, J. A. V. D.; COPLIEN, J. O. Using Design Patterns to Build a Framework for Multimedia Networking. **Bell Labs Technical Journal**, v.2, n.1, p.166-187, February. 1997.

CAETANO, C. **CVS: Controle de Versões e Desenvolvimento Colaborativo de Software.** 1.ed. São Paulo: Novatec, 2004. 141p.

CAGNIN, M. I. **PARFAIT: uma contribuição para a Reengenharia de Software baseada em Linguagem de Padrões e *Frameworks***. 2005. Tese (Doutorado em Ciência da Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2005.

CAGNIN, M. I.; MALDONADO, J. C.; BRAGA, R. T. V.; GERMANO, F.; PENTEADO, R. Uma ferramenta de apoio ao Controle de Versão das Aplicações criadas por um *Framework*. In: **XXX Conferência Latino – Americana de Informática (CLEI2004)**, Arequipa – Peru, p.414-425, 2004a.

CAGNIN, M. I.; MALDONADO, J. C.; MASIERO, P. C.; PENTEADO, R. D.; BRAGA, R. T. An Evolution Process for Application Frameworks. In: **Workshop de Manutenção Moderna de Software**, em conjunto com o XVIII Simpósio Brasileiro de Engenharia de Software, Brasília, DF, CD-ROM, 8p., 2004b.

CAMARGO, V. V.; MASIERO, P. C. *Frameworks* Orientados a Aspectos. In: **Anais 19º Simpósio Brasileiro de Engenharia de Software (SBES'2005)**, Uberlândia – MG, Brasil, Outubro, 2005.

CAMARGO, V. V. ***Frameworks* Transversais: Definições, Classificações, Arquitetura e Utilização em um Processo de Desenvolvimento de Software.** Tese (Doutorado em Ciência da Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2006.

CEDERQVIST, P. **Version Management with CVS**. Official manual for CVS. 1993. Disponível em: <http://www.cvshome.org/docs/manual>. Acesso em: 10 mar. 2006.

COLLINS – SUSSMAN, B.; FITZPATRICK, B.W.; PILATO, C. M. **Version Control with Subversion: for Subversion 1.0**. O'Reilly & Associates, Sebastopol, Califórnia, June. 2004. Livro online. Disponível em: <http://svnbook.red-bean.com>. Acesso em: 16 mar. 2006.

CONRADI, R.; WESTFECHTEL, B. Version Models for Software Configuration Management. **ACM Computing Surveys (CSUR)**, v.30, n.2, p. 232-282, June. 1998.

CONSTANTINIDES C. A.; SKOTINIOTIS T. Reasoning about a Classification of Crosscutting Concerns in Object-Oriented Systems. In: **2th Workshop on Aspect Oriented Software Development of SIG (Software Interesting Group), German Informatics Society.** Bonn, Alemanha, February, 2002.

COUTO, C. F. M.; VALENTE, M. T. O.; BIGONHA, R. S. Um Arcabouço Orientado por Aspectos para Implementação Automatizada de Persistência. In: **Anais do 2º. Workshop Brasileiro de Desenvolvimento de Software Orientado a Aspectos (WASP'05),** evento satélite do XIX SBES, Uberlândia, MG, Brasil, Outubro, 2005.

DEITEL, H. M.; DEITEL, P. J. **Java: Como Programar.** 4ª ed. Porto Alegre: Bookman. 2003

FAYAD, M. E.; SCHMIDT, D. C. Object-Oriented Application Frameworks. **Communications of the ACM**, v.40, n.10, p.32-38, October. 1997.

FAYAD, M. E.; SCHIMIDT, D. C.; JOHNSON, R. **Building Application Frameworks: Object-oriented Foundations of Framework Design.** John Wiley & Sons, 1999.

FIORINI, S. T. **Arquitetura para Reutilização de Processos de Software**. 2001. Tese (Doutorado em Informática) – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, 2001.

FIREBIRD PROJECT. **Firebird Database.** Disponível em: < http://firebird.sourceforge.net>. Acesso em: 10 jul. 2006.

FUTRELL, R. T.; SHAFER, L. I.; SHAFER, D. F; **Quality Software Project Management.** 1st ed. Prentice Hall. 2002. 1680p.

GROTT, M. C. **Reutilização de soluções com Patterns e Frameworks na Camada de Negócio.** 2003. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, 2003a.

GROTT, M. C.; SOUSA, F. C.; HUGO, M. Reutilização de soluções com Patterns e *Frameworks* na Camada de Negócio. In: **XIII SEMINCO – Seminário de Computação**, 1, 2003, Blumenau, XIII SEMINCO. Blumenau: Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau, 2003b. 12p.

HANENBERG, S.; HIRSCHFELD, A.; UNLAND, R.; KAWAMURA, K. Applying Aspect-Oriented Composition to Framework Development – A Case Study. In: **Proceedings of the 1st International Workshop on Foundations of Unanticipated Software Evolution,** Barcelona, Spain, March 28, 2004.

HUANG, M., WANG, C., ZHANG, L. Towards a Reusable and Generic Aspect Library. In: **Proceedings of AOSDSEC'04 (AOSD Technology for Application-Level Security).** Workshop of the Aspect Oriented Software Development Conference, Lancaster, UK, March, 23, 2004.

IBM. **Rational Software**. Disponível em: <http://www-306.ibm.com/software/rational/>. Acesso em: 15 jun. 2006.

JOHNSON, R. E. Frameworks = (Patterns + Components). **Communication of the ACM**, v.40, n.10, p.39-42, October. 1997.

JOHNSON, R. E.; FOOTE, B. Designing Reusable Classes. **Journal of Object – Oriented Programming**, v.1, n.2, p. 22-35, June-July. 1988.

KICZALES, G. Aspect-Oriented Programming. **ACM Computing Surveys**, v.28, n.4, 154p. 1996.

KICZALES, G.; LAMPING, J.; MENDHEKAR, A.; MAEDA, C.; LOPES, C.; LOINGTIER, J. and IRVING, J. 1997. "Aspect Oriented Programming". In: **Proceedings of 11 ECOOP**, p. 220-242, 1997.

LACERDA, D. S.; BARTH, F. J.; GOMI, E. S. WebRCS - Um Sistema de Gerenciamento de Documentos via Web. In: **Simpósio Internacional de Iniciação Cientifica da Universidade de São Paulo,** 9, 2001, São Paulo. 9º SIICUSP; XVII CICTE: resumos. São Paulo: USP, 2001. 224p.

LUCENA, F. N. **Subversion: uma Introdução.** Goiânia, 43p., 2005. (Relatório Técnico) – Instituto de Informática, Universidade Federal de Goiás, 2005.

MACHADO, R. **A hora da GCS.** Revista Tema, Brasília, n.175, p.52-55, Setembro-Outubro. 2002.

PINTO, M.; FUENTES, L.; FAYAD, M. E; TROYA, J. M. Separation of Coordination in a Dynamic Aspect Oriented Framework. In: **Proceeding of the 1st International Conference on Aspect-Oriented Software Development,** April 2002.

PREE, W. **Hot-Spot-Driven Development** in M. Fayad, R. Johnson, D. Schmidt. Building Application Frameworks: Object-Oriented Foundations of Framework Design, John Willey and Sons, p. 379–393, 1999.

PRESSMAN, R. S. **Software Engineering: a practitioner's approach**. 5th ed. McGraw-Hill, 2001.

RASHID, A.; CHITCHYAN, R. Persistence as an Aspect. In: **Proceedings of the 2nd International Conference on Aspect Oriented Software Development (AOSD),** Boston–USA, March, 2003.

ROBERTS, D.; JOHNSON, R. Evolving Frameworks: a Pattern Language for Developing Object-Oriented Frameworks. In: **3rd Conference on Pattern Language Programming**, 1996, Montecillio: University of Illinois, 1996. 16p.

RUEDA, R.; CARLOS, V. A. **Suporte à Gerência de Configuração.** 2003. 12f. Trabalho de Conclusão de Disciplina (Engenharia de Software) – Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis, 2003.

SILVA, R. P. **Suporte ao Desenvolvimento e uso de *Frameworks* e Componentes**. 2000. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, 2000.

SILVA, F. A. **RCS e CVS: Ferramentas para Versionamento de Arquivos.** Porto Alegre, RS, 2002. Disponível em: <http://www.inf.ufrgs.br/~clesio/cmp151/cmp15120021/artigo_ fabricio.pdf>. Acesso em: 10 mar. 2006.

SILVA, S. R. Q. M. **Controle de Versões – Um apoio à Edição Colaborativa na Web**. 2005. Dissertação (Mestrado em Ciência da Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2005.

SOARES, M. D. **Gerenciamento de Versões de Páginas Web.** 2000. Dissertação (Mestrado em Ciência da Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2000.

SOARES, S.; LAUREANO, E.; BORBA, P. Implementing Distribution and Persistence Aspects with AspectJ. In: **Proceedings the 17th ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA),** p. 174-190, November, 2002.

SOMMERVILLE, I. **Engenharia de Software**. 6ª ed., São Paulo: Addison-Wesley, 2004.

SOUZA, M. V. B**. Estudo Comparativo entre *Frameworks* Java para Construção de Aplicações Web.** 2004. Trabalho de Conclusão de Curso, (Graduação em Ciência da Computação) – Faculdade de Informática, Universidade Federal de Santa Maria, 2004.

SUN MICROSYSTEMS. **Javadoc.** Disponível em: <http://java.sun.com/j2se/javadoc/>. Acesso em: 15 jul. 2006.

SUN MICROSYSTEMS. **JavaServer Pages Technology.** Disponível em: <http://java.sun.com/products/jsp/index.jsp>. Acesso em: 14 ago. 2006.

SUN MICROSYSTEMS. **Servlet Technology.** Disponível em:<http://java.sun.com/products/servlet/>. Acesso em: 14 ago. 2006.

SCHMIDT, D. C.; BUSCHMANN, F. Patterns, Frameworks, and Middleware: Their Synergistic Relationships. In: **25th International Conference on Software Engineering,** 2003, Portland, Oregon, 2003. p.694-704.

SHAH, V.; HILL, V. An Aspect-Oriented Security Framework: Lessons Learned. In: **Proceedings of AOSDSEC'04 (AOSD Technology for Application-Level Security).** Workshop of the Aspect Oriented Software Development Conference, Lancaster, UK, March, 23, 2004.

TALIGENT, I. **Leveraging Object-Oriented Frameworks**. 16p. 1993. Taligent White Papers. Disponível em : <http://lhcb-comp.web.cern.ch/lhcb - comp/Components/postscript/leveragingoo.pdf>. Acesso em: 15 fev. 2006.

TALIGENT, I. **Building Object – Oriented Frameworks**. 23p. 1997. Taligent White Papers. Disponível em: <http://lhcb-comp.web.cern.ch/lhcb-comp/Components/postscript/buildingoo.pdf>. Acesso em: 15 fev. 2006.

TIBERTI, A. J. **Desenvolvimento de Software de apoio ao Projeto de Arranjo Físico de Fábrica baseado em um *Framework* Orientado a Objeto.** 2003. Tese (Doutorado em Engenharia Mecânica) – Escola de Engenharia de São Carlos, Universidade de São Paulo. 2003.

TICHY, W. F. RCS – A System for Version Control. **Software - Practice and Experience**, v.15, n.7, p.637-654, 1985.

TICHY, W. F. Tools for Software Configuration Management. In: **International Workshop on Software Version and Configuration Control**, Grassau, 1988. p.1-20.

TODD, N.; SZOLKOWSKI, M**. JavaServer Pages: o Guia do Desenvolvedor**. 1 ed. Rio de Janeiro: Campus. 2003. 621p.

VANHAUTE, B.; WIN, B.; DECKER, B. Building Frameworks in AspectJ. In: **Proc. of the 15th European Conference on Object-Oriented Programming (ECOOP),** Separation of Concerns Workshop. p. 1-6, June, 2001.

WESTFECHTEL, B. Software Configuration Management. In: **11th International Workshop on Software Configuration Management (SCM-11)**, Portland, Oregon, 2003 p.1-3.

APÊNDICE A – Documento de Requisitos

A – Visão Geral da Ferramenta

A ferramenta trata basicamente do gerenciamento do cadastro das versões do *framework* e das versões das aplicações vinculadas aos *frameworks*. Para isso, será definido um repositório contendo os dados de cada versão. Além disso, a ferramenta deve armazenar os requisitos funcionais e não funcionais implementados em cada versão do *framework* e deve manter um registro dos requisitos não cobertos por cada versão do *framework* em um Histórico de Requisitos, a fim de auxiliar na decisão de evoluir ou não o *framework* com o apoio do PREF (Processo de Evolução de *Frameworks* de Aplicação) (CAGNIN *et. al*., 2004; CAGNIN, 2005).

Quando um requisito não é considerado essencial ao *framework*, este é cadastrado no Histórico de Requisitos e a situação do requisito é atribuída inicialmente como "pendente", ou seja, o requisito não é fornecido pelo *framework*. Por outro lado, se um requisito for considerado essencial ao *framework*, inicia-se a sua implementação no *framework* e a sua situação no Histórico de Requisitos será alterada para o estado "sendo atendido". Assim que o requisito for completamente implementado no *framework*, sua situação será atualizada no Histórico de Requisitos, sendo alterada para o estado "atendido". O histórico contém várias informações como: a palavra chave/identificador de cada requisito, a descrição de cada requisito, a solução de implementação adotada no sistema (quando o requisito não é fornecido pelo *framework* e é implementado manualmente no sistema gerado a partir da instanciação do *framework*), a razão pela qual o requisito não é considerado essencial ao domínio do *framework*, o nome do sistema em que o requisito foi incorporado, entre outros. Todas as consultas ao Histórico de Requisitos são baseadas nestas informações, que são necessárias e exigidas pelo PREF.

B – Requisitos Funcionais

B1 – Cadastro e Manutenção Diversos

1. A ferramenta deve permitir a inclusão, alteração e exclusão dos *frameworks,* contendo os seguintes itens de informação: código de identificação do *framework*, nome do *framework*, proprietário do *framework*, domínio do *framework*, linguagem no qual o *framework* foi implementado e descrição do *framework*.
2. A ferramenta deve permitir a inclusão, alteração e exclusão das versões do *framework* e do código fonte de cada versão, contendo os respectivos itens de informação: código de identificação da versão do *framework*, código de identificação do *framework* que gerou a versão, número da versão do *framework* e data da criação da versão.
3. A ferramenta deve permitir a inclusão, alteração e exclusão do domínio em que o *framework* foi projetado, com os seguintes itens de informação: código de identificação do domínio, nome (domínio coberto) e descrição.
4. A ferramenta deve permitir a inclusão, alteração e exclusão das aplicações vinculadas ao *framework*, contendo os seguintes itens de informação: código de identificação da aplicação, nome da aplicação, proprietário da aplicação e descrição.
5. A ferramenta deve permitir a inclusão, alteração e exclusão das versões das aplicações vinculadas ao *framework*, contendo os respectivos itens de informação: código de identificação da versão da aplicação, código da aplicação, número da versão da aplicação, código de identificação da versão do *framework* vinculado a versão da aplicação, número da versão do *framework*, data da criação da versão e descrição.
6. A ferramenta deve permitir a inclusão, alteração e exclusão dos requisitos (funcionais e não funcionais) não implementados/atendidos por cada versão do *framework* – quando não estão presentes no *framework* e são implementados manualmente em

alguma aplicação. Os seguintes itens de informação são necessários: código de identificação de cada requisito, palavra-chave associada ao requisito, solução de projeto implementada na aplicação, razão pela qual o requisito não é essencial, nome da aplicação em que o requisito foi implementado, tipo de manutenção realizada, tipo de requisito (funcional ou não funcional), a situação do requisito em relação ao *framework* (estado "pendente" – indica que o requisito ainda não foi incorporado ao *framework*, estado "sendo atendido" – indica que o requisito está sendo implementado e estado "atendido" – indica que o requisito foi incorporado ao *framework*), a versão do *framework* em que o requisito foi incorporado e a descrição do requisito.

7. A ferramenta deve permitir a inclusão, alteração e exclusão dos requisitos (funcionais e não funcionais) implementados/atendidos em cada versão do *framework*, contendo o código de identificação de cada requisito, a palavra-chave associada ao requisito, a versão do *framework* em que o requisito foi incorporado e a descrição.
8. A ferramenta dever permitir a inclusão, alteração e exclusão das palavras-chaves associadas aos requisitos (funcionais ou não funcionais), implementados/atendidos e não implementados/atendidos por cada versão do *framework*. Os seguintes itens de informação serão necessários: código de identificação da palavra-chave e a palavra-chave.A ferramenta deve permitir a inclusão, alteração e exclusão dos usuários (administrador do sistema, proprietário do *framework*, proprietário da aplicação e usuário comum), contendo os seguintes itens de informação: código de identificação do usuário, nome do usuário, sobrenome do usuário, e-mail, nome de identificação do usuário *(username)*, senha de acesso *(password)* e o papel desempenhado por ele (tipo de usuário).

B2. Controle de Consistência das Informações

9. A ferramenta deve, se houver a tentativa de inclusão de um dado já existente, informar ao usuário sobre a existência desse respectivo dado. Caso o usuário prossiga com a operação de inclusão, a ferramenta deve informar sobre essa alteração.
10. A ferramenta deve enviar mensagens de erro caso informações incompletas sejam incluídas. Tais mensagens interrogam o usuário se deseja cancelar a operação de inclusão, completar as informações incompletas ou concluir a inclusão assim mesmo.
11. A ferramenta deve enviar mensagens de erro caso ocorra algum problema na alteração de um dado. Tais mensagens interrogam o usuário se deseja cancelar a operação de alteração ou prosseguir assim mesmo.
12. A ferramenta deve enviar mensagens de erro caso ocorra algum problema na exclusão de um dado. Tais mensagens interrogam o usuário se deseja cancelar a operação de exclusão ou prosseguir assim mesmo.

B3 – Consultas Gerais

13. A ferramenta deve permitir o processamento de consulta *on-line* das informações relativas ao *framework*. Nessas informações devem constar: código de identificação do *framework*, nome, proprietário, domínio, linguagem no qual o *framework* foi implementado e descrição.
14. A ferramenta deve permitir o processamento de consulta *on-line* das informações relativas as versões do *framework*. Nessas informações devem constar: código da versão do *framework*, código do *framework*, número da versão do *framework* e data da criação da versão.

15. A ferramenta deve permitir o processamento de consulta *on-line* dos requisitos não cobertos pelo *framework* e que foram implementados em uma determinada versão de uma aplicação. Nessa consulta deve conter: código do requisito, palavra-chave associada ao requisito, solução de projeto implementada na aplicação, razão pela qual o requisito não é essencial, nome da aplicação em que o requisito foi implementado, tipo de manutenção, tipo de requisito (funcional ou não funcional) e descrição.
16. A ferramenta deve permitir o processamento de consulta *on-line* de uma solução de projeto de um determinado requisito, adotado durante a implementação do mesmo em uma aplicação.
17. A ferramenta deve permitir o processamento de consulta *on-line* de informações referente às vinculadas a cada versão do *framework*. Essa consulta deve conter: código de identificação da aplicação, nome da aplicação, proprietário da aplicação e descrição.
18. A ferramenta deve permitir o processamento de consulta *on-line* de informações referente às versões das aplicações vinculadas a cada versão do *framework*. Essa consulta deve conter: código de identificação da versão da aplicação, código da aplicação, versão da aplicação, código da versão do *framework*, versão do *framework*, data da criação da versão e descrição.
19. A ferramenta deve permitir o processamento de consulta *on-line* dos requisitos implementados em uma determinada versão do *framework*. Na consulta deve constar: código de identificação do requisito, palavra-chave associada ao requisito, situação do requisito em relação ao *framework*, código da versão do *framework* em que o requisito foi implementado e descrição.

20. A ferramenta deve permitir o processamento de consulta *on-line* no Histórico de Requisitos para exibir uma lista dos requisitos cobertos e não cobertos por cada versão do *framework*.
21. A ferramenta deve permitir o processamento de consulta *on-line* no Histórico de Requisitos para analisar se um requisito é considerado essencial ao domínio do *framework*. Um requisito é considerado essencial ao domínio do *framework* se existir pelo menos duas situações em que o requisito pode ocorrer novamente e se existem no mínimo mais duas ocorrências do requisito em questão, cuja situação esteja no estado "pendente". Se esse for o caso, o requisito provavelmente é essencial ao domínio do *framework* e deve ser incorporado. Para auxiliar na análise dos requisitos não cobertos pelo *framework* os seguintes itens de informação são necessários: palavra-chave de identificação do requisito e situação em que o requisito pode ocorrer novamente.

C – Requisitos Não Funcionais

C1 – Controle de Acesso

22. A ferramenta deve possuir um controle de acesso para os diferentes tipos de usuários: administrador do sistema, proprietário do *framework*, proprietário da aplicação e usuário comum.

C2 – Segurança de Acesso

23. A ferramenta deve manter a segurança de acesso, garantido que, os cadastros não sejam alterados por parte de usuários não autorizados.

C3 – Confiabilidade

24. A ferramenta deve ser tolerante à falhas, tendo a capacidade de recuperar todos os dados perdidos durante a última operação, mantendo a consistência e a integridade dos dados.

C4 – Eficiência

25. O tempo de resposta para as operações de inserção, alteração e exclusão não deve exceder a cinco segundos.
26. O tempo de processamento de uma consulta *on-line* não deve exceder a cinco segundos.

C5 – Requisitos Mínimos de Hardware/Software

27. **Máquina Cliente** – a ferramenta deve ser executada em microcomputadores Pentium III 450mHz ou superior, com memória RAM mínima de 64Mbytes e na plataforma Windows.
28. **Máquina Servidor Web** – a ferramenta dever ser executada preferencialmente no servidor Web Apache e na plataforma Windows.
29. A ferramenta deve necessariamente ser capaz de armazenar os dados em uma base de dados Firebird.

APÊNDICE B – Descrição dos casos de uso

Caso de Uso: Efetuar Login

- **Atores:** Administrador do Sistema, Proprietário do *Framework*, Proprietário da Aplicação e Usuário Comum.
- **Descrição:** Este caso de uso é responsável pela permissão de acesso as funcionalidades oferecidas pelo sistema.
- **Dados_login:** usuário + senha
- **Curso Normal:**
  1. Usuário informa nome de usuário (username) e senha (password) válidos
  2. Sistema mostra a página correspondente ao tipo de usuário
- **Curso Alternativo 1:**
  2. Usuário informa nome de usuário (username) e/ou senha (password) inválidos
     2.1. Sistema emite mensagem “Nome de usuário e/ou senha inválidos”
     2.2. Retorna para o passo 1

Caso de Uso: Gerenciar Pré-Usuários

- **Ator:** Administrador do Sistema
- **Descrição:** Este caso de uso é responsável pelo gerenciamento das pessoas interessadas na utilização da ferramentausuários do sistema.
- **Dados_usuários** = nome + sobrenome + e-mail + username + password + user_type
- **Curso Normal:**

1. Usuário informa o e-mail
2. Sistema verifica que não existe usuário associado ao e-mail informado
3. Usuário informa o restante dos dados.
4. Sistema cria um código de identificação e cria uma instância de usuário
5. Sistema emite mensagem “Operação realizada com sucesso”

- **Curso Alternativo:**

2. Sistema verifica que existe usuário associado ao e-mail informado

2.1. Sistema emite mensagem “Cadastro já existe”

2.2. Sistema exibe dados do usuário

2.3. Abandonar caso de uso

Caso de Uso: Gerenciar Usuários

- **Ator:** Administrador do Sistema
- **Descrição:** Este caso de uso é responsável por gerenciar os diferentes tipos de usuários do sistema.
- **Dados_usuários** = nome + sobrenome + e-mail + username + password + user_type
- **Curso Normal:**

1. Usuário informa o e-mail
2. Sistema verifica que não existe usuário associado ao e-mail informado
3. Usuário informa o restante dos dados
4. Sistema cria um código de identificação e cria uma instância de usuário
5. Sistema emite mensagem “Operação realizada com sucesso”

- **Curso Alternativo:**

  2. Sistema verifica que existe usuário associado ao e-mail informado

     2.1. Sistema emite mensagem “Cadastro já existe”

     2.2. Sistema exibe dados do usuário

     2.3. Abandonar caso de uso

Caso de Uso: Gerenciar *Frameworks*

- **Ator:** Administrador do Sistema e Proprietário do *Framework*
- **Descrição:** Este caso de uso é responsável por cadastrar os *frameworks.*
- **Dados*_framework*** **=** nome + proprietário + domínio*_framework* + linguagem_implementação
- **Curso Normal:**

  1. Usuário informa Dados*_framework*
  2. Sistema verifica que não existe *framework* associado ao nome informado
  3. Sistema verifica que o domínio do *framework* informado é válido
  4. Sistema cria um código de identificação e cria uma instância de *framework*
  5. Sistema emite mensagem “Operação realizada com sucesso”

- **Curso Alternativo 1:**

  2. Sistema verifica que existe *framework* associado ao nome informado

     2.1. Sistema emite mensagem “Cadastro já existe”

     2.2. Sistema exibe dados do *framework*

     2.3. Abandonar caso de uso

- **Curso Alternativo 2:**

  3. Sistema verifica que o domínio do *framework* é invalido

3.1. Sistema emite mensagem “Domínio inválido”

3.2. Abandonar caso de uso

Caso de Uso: Gerenciar Versões de *Frameworks*

- **Atores:** Administrador do Sistema e Proprietário do *Framework*
- **Descrição:** Este caso de uso é responsável pelo cadastro das versões de *frameworks.*
- **Dados_versão*_framework* =** código + nome + proprietário + domínio*_framework* + linguagem_implementação + versão*_framework* + data_criação
- **Curso Normal:**
  1. Usuário informa Dados_versão*_framework*
  2. Sistema verifica que não existe versão de *framework* associado ao nome informado
  3. Sistema verifica que o domínio da versão do *framework* informado é válido
  4. Sistema cria um código de identificação e cria uma instância da versão de *framework*
  5. Sistema emite mensagem “Operação realizada com sucesso”
- **Curso Alternativo 1:**

  2. Sistema verifica que existe versão de *framework* associado ao nome informado

  2.1. Sistema emite mensagem “Cadastro já existe”

  2.2. Sistema exibe dados da versão do *framework*

  2.3. Abandonar caso de uso
- **Curso Alternativo 2:**
  3. Sistema verifica que o domínio da versão do *framework* é invalido

3.1. Sistema emite mensagem “Domínio inválido”

3.2. Abandonar caso de uso

Caso de Uso: Gerenciar Domínio do *Framework*

- **Atores:** Administrador do Sistema e Proprietário do *Framework*
- **Descrição:** Este caso de uso é responsável por cadastrar o domínio em que o *framework* foi projetado.
- **Dados_domínio =** nome_domínio/tipo + descrição
- **Curso Normal:**
  1. Usuário informa Dados_domínio
  2. Sistema verifica que não existe domínio associado ao nome informado
  3. Sistema cria um código de identificação e cria uma instância de domínio
  4. Sistema emite mensagem “Operação realizada com sucesso”
- **Curso Alternativo:**
  2. Sistema verifica que existe domínio associado ao nome informado
     2.1. Sistema emite mensagem “Cadastro já existe”
     2.2. Abandonar caso de uso

Caso de Uso: Gerenciar Funcionalidades Implementadas em cada Versão do *Framework*

- **Ator:** Administrador do Sistema, Proprietário do *Framework*
- **Descrição:** Este caso de uso é responsável por cadastrar os requisitos (funcionais ou não funcionais) cobertos pelo *framework*, ou seja, implementados/atendidos em cada versão do *framework*.

- **Dados_requisitos** = palavra_chave + situação_requisito + versão_*framework* + descrição_requisito
- **Curso Normal:**
  1. Usuário informa Dados_requisitos
  2. Sistema verifica que não existe requisito associado aos dados fornecidos
  3. Sistema cria um código de identificação e cria uma instância de requisito
  4. Sistema emite mensagem “Operação realizada com sucesso”
- **Curso Alternativo:**
  2. Sistema verifica que existe requisito associado aos dados fornecidos
     2.1. Sistema emite mensagem “Cadastro já existe”
     2.2. Sistema exibe dados do requisito
     2.3. Abandonar caso de uso

Caso de Uso: Gerenciar Funcionalidades não Implementadas no *Framework*

- **Atores:** Administrador do Sistema, Proprietário do *Framework* e Proprietário da Aplicação
- **Descrição:** Este caso de uso é responsável por cadastrar os requisitos (funcionais ou não funcionais) não cobertos pelo *framework* e que foram implementados manualmente em uma aplicação.
- **Dados_requisitos** = palavra_chave + solução_projeto + razão_implementação + nome_aplicação + tipo_manutenção + tipo_requisito
- **Curso Normal:**
  1. Usuário informa Dados_requisitos
  2. Sistema verifica que não existe requisito associado aos dados fornecidos

3. Sistema cria um código de identificação e cria uma instância de requisito
4. Sistema emite mensagem“Operação realizada com sucesso”

- **Curso Alternativo:**

1. Sistema verifica que existe requisito associado aos dados fornecidos

2.1. Sistema emite mensagem “Cadastro já existe”

2.2. Sistema exibe dados do requisito

2.3. Abandonar caso de uso

Caso de Uso: Gerenciar Aplicações Vinculadas ao *Framework*

- **Ator:** Administrador do Sistema, Proprietário do *Framework* e Proprietário da Aplicação
- **Descrição:** Este caso de uso é responsável por cadastrar as aplicações vinculadas ao *framework*.
- **Dados_aplicação =** nome_aplicação + proprietário_aplicação + versão_*framework* + descrição
- **Curso Normal:**

1. Usuário informa Dados_aplicação
2. Sistema verifica que não existe aplicação associada aos dados fornecidos
3. Sistema cria um código de identificação e cria uma instância de aplicação
4. Sistema emite mensagem “Operação realizada com sucesso”

- **Curso Alternativo:**

2. Sistema verifica que existe aplicação associada ao dados fornecidos

2.1. Sistema emite mensagem “Cadastro já existe”

2.2. Sistema exibe dados da aplicação

2.3. Abandonar caso de uso

Caso de Uso: Gerenciar Versões das Aplicações Vinculadas ao *Framework*

- **Ator:** Administrador do Sistema, Proprietário do *Framework* e Proprietário da Aplicação
- **Descrição:** Este caso de uso é responsável pelo cadastro das versões das aplicações vinculadas ao *framework*.
- **Dados_versão_aplicação =** código_aplicação + nome_aplicação + proprietário_aplicação + versão_*framework* + versão_aplicação + data_criação + descrição
- **Curso Normal:**
  1. Usuário informa Dados_versão_aplicação
  2. Sistema verifica que não existe versão da aplicação associada aos dados fornecidos
  3. Sistema cria um código de identificação e cria uma instância de versão da aplicação
  4. Sistema emite mensagem "Operação realizada com sucesso"
- **Curso Alternativo:**
  2. Sistema verifica que existe versão da aplicação associada ao dados fornecidos
     2.1. Sistema emite mensagem "Cadastro já existe"
     2.2. Sistema exibe dados da versão da aplicação
     2.3. Abandonar caso de uso

Caso de Uso: Consultar Informações do *Framework*

- **Atores:** Administrador do Sistema, Proprietário do *Framework*, Proprietário da Aplicação e Usuário Comum
- **Descrição:** Este caso de uso é responsável por consultar informações de um determinado *framework*.
- Dados_consulta = nome
- Curso Normal:
  1. Usuário informa Dados_consulta
  2. Sistema seleciona proprietário_*framework*, domínio_*framework*, linguagem_ implementação, número_versão, data_criação da classe *Framework* com nome = nome fornecido
  3. Sistema exibe dados selecionados
- **Curso Alternativo:**
  2. Sistema não encontra *framework* com nome = nome fornecido
     2.1. Sistema emite mensagem "Cadastro não existe"
     2.2. Abandonar caso de uso

Caso de Uso: Consultar Informações das Aplicações Vinculadas ao *Framework*

- **Atores:** Administrador do Sistema, Proprietário do *Framework*, Proprietário da Aplicação e Usuário Comum
- **Descrição:** Este caso de uso é responsável por consultar informações das aplicações vinculadas ao *framework*
- **Dados_consulta =** nome

- **Curso Normal:**
  1. Usuário informa Dados_consulta
  2. Sistema seleciona código, proprietário, versão_*framework* e descrição da classe Aplicação com nome = nome fornecido
  3. Sistema exibe dados selecionados
- **Curso Alternativo:**
  2. Sistema não encontra aplicação com nome = nome fornecido
     2.1. Sistema emite mensagem "Cadastro não existe"
     2.2. Abandonar caso de uso

Caso de Uso: Consultar Solução de Projeto de um Requisito Implementado na Aplicação

- **Atores:** Administrador do Sistema, Proprietário do *Framework*, Proprietário da Aplicação e Usuário Comum
- **Descrição:** Este caso de uso é responsável por consultar informações sobre uma solução de projeto de um determinado requisito, adotada na implementação em alguma versão de uma aplicação
- **Dados_consulta:** nome/descrição_requisito
- **Curso Normal:**
  1. Usuário informa Dados_consulta
  2. Sistema seleciona código, palavra_chave, solução_projeto, razão_implementação, nome_aplicação, tipo_manutenção e tipo_requisito com nome/descrição do requisito = descrição fornecida
  3. Sistema exibe dados selecionados

Caso de Uso: Consultar Histórico de Requisitos

Inclui: Consultar Requisitos Cobertos pelo *Framework*

- **Ator:** Administrador do Sistema, Proprietário do *Framework*, Proprietário da Aplicação e Usuário Comum
- **Descrição:** Este caso de uso é responsável por consultar no Histórico de Requisitos os requisitos cobertos pelo *framework*, ou seja, implementados/atendidos por cada versão do *framework*.
- **Dados_requisito** = palavra_chave
- **Curso Normal:**
    1. Usuário informa Dados_requisito
    2. Sistema seleciona código, situação_requisito, código_versão_*framework* e descrição_requisito da classe Requisitos Cobertos pelo *Framework* com palavra_chave = palavra_chave informada
    3. Sistema exibe dados selecionados

Inclui: Consultar Requisitos não Cobertos pelo *Framework*

- **Ator:** Administrador do Sistema, Proprietário do *Framework*, Proprietário da Aplicação e Usuário Comum
- **Descrição:** Este caso de uso é responsável por consultar no Histórico de Requisitos os requisitos não cobertos pelo *framework* para analisar se um determinado requisito é considerado essencial ao domínio do *framework*.
- **Dados_requisito** = palavra_chave
- **Curso Normal:**
    1. Usuário informa Dados_requisito

2. Sistema seleciona código, solução_projeto, razão_implementação, nome_aplicação, tipo_manutenção, tipo_requisito e descrição_requisito da classe Requisitos Não Cobertos pelo *Framework* com palavra_chave = palavra_chave informada
3. Sistema exibe dados selecionados

APÊNDICE C – Script de Criação da Base de Dados da Ferramenta

```
CREATE DATABASE IF NOT EXISTS TOOLBASE;

CREATE TABLE USERS (
        USERS_ID       INTEGER NOT NULL AUTO_INCREMENT,
        FIRSTNAME    VARCHAR(30),
        LASTNAME     VARCHAR(40),
        EMAIL            VARCHAR(40) NOT NULL,
        USERNAME     VARCHAR(30) NOT NULL,
        PASSWORDS   VARCHAR(30) NOT NULL,
        USER_TYPE     VARCHAR(25) NOT NULL
);
ALTER TABLE USERS ADD CONSTRAINT PK_USERS PRIMARY KEY (USERS_ID);


CREATE TABLE PRE_USERS (
        PREUSERS_ID   INTEGER NOT NULL AUTO_INCREMENT,
        FIRSTNAME      VARCHAR(30),
        LASTNAME       VARCHAR(40),
        EMAIL             VARCHAR(40) NOT NULL,
        USERNAME       VARCHAR(30) NOT NULL,
        PASSWORDS     VARCHAR(30) NOT NULL,
        USER_TYPE       VARCHAR(25) NOT NULL,
        PURPOSE          VARCHAR(200) NOT NULL
);
ALTER TABLE PRE_USERS ADD CONSTRAINT PK_PRE_USERS
        PRIMARY KEY (PREUSERS_ID);


CREATE TABLE WORK_ROLES (
        ROLES_ID  INTEGER NOT NULL,
        ROLES     VARCHAR(30) NOT NULL
);
ALTER TABLE WORK_ROLES ADD CONSTRAINT PK_WORK_USERS
        PRIMARY   KEY (ROLES_ID);


CREATE TABLE FRAMEWORK (
        FRAM_ID            INTEGER NOT NULL AUTO_INCREMENT,
        NAME               VARCHAR(30) NOT NULL,
        OWNER              VARCHAR(40) NOT NULL,
        DOMAIN_TYPE   VARCHAR(30) NOT NULL,
        LANGUAGE       VARCHAR(30) NOT NULL,
        FRAM_TYPE      VARCHAR(35) NOT NULL,
        DESCRIPTION    VARCHAR(200),
);
ALTER TABLE FRAMEWORK ADD CONSTRAINT PK_FRAMEWORK
        PRIMARY KEY (FRAM_ID);
```

```
CREATE TABLE FRAMEWORK_VERSION (
        FRAMVERSION_ID    INTEGER NOT NULL AUTO_INCREMENT,
        FRAM_ID           VARCHAR(30) NOT NULL,
        FRAM_VERSION      VARCHAR(5) NOT NULL,
        CREATE_DATE       DATE NOT NULL
);
ALTER TABLE FRAMEWORK_VERSION
        ADD CONSTRAINT    PK_FRAMEWORK_VERSION
              PRIMARY KEY (FRAMVERSION__ID);
ALTER TABLE FRAMEWORK_VERSION
        ADD CONSTRAINT FK_FRAMEWORK_VERSION_1
              FOREIGN KEY (FRAM_ID)
         REFERENCES FRAMEWORK (FRAM_ID);


CREATE TABLE DOMAINS (
        DOMAINS_ID        INTEGER NOT NULL AUTO_INCREMENT,
        DOMAIN_TYPE       VARCHAR(30) NOT NULL,
        DESCRIPTION       VARCHAR(200) NOT NULL
);
ALTER TABLE DOMAINS ADD CONSTRAINT PK_DOMAINS
        PRIMARY KEY (DOMAINS_ID);


CREATE TABLE REQUIREMENTS_COVERED (
        REQ_ID            INTEGER NOT NULL AUTO_INCREMENT,
        KEYWORDS          VARCHAR(40) NOT NULL,
        FRAM_ID           INTEGER NOT NULL,
        DESCRIPTION       VARCHAR(200) NOT NULL
);
ALTER TABLE REQUIREMENTS_COVERED ADD
        CONSTRAINT PK_REQUIREMENTS_COVERED PRIMARY KEY (REQ_ID);


CREATE TABLE REQ_INCOVERED (
        REQ_ID            INTEGER NOT NULL AUTO_INCREMENT,
        KEYWORDS          VARCHAR(40) NOT NULL,
        PROJ_SOL          VARCHAR(200) NOT NULL,
        IMP_REASON        VARCHAR(200) NOT NULL,
        APP_NAME          VARCHAR(30) NOT NULL,
        MAIN_TYPE         VARCHAR(30) NOT NULL,
        REQ_TYPE          VARCHAR(30) NOT NULL,
        STATUS            VARCHAR(20) NOT NULL,
        FRAMVERSION_ID    VARCHAR(4) NULL,
        DESCRIPTION       VARCHAR(250) NOT NULL
);
ALTER TABLE REQ_INCOVERED ADD CONSTRAINT PK_REQ_INCOVERED
PRIMARY KEY (REQ_ID);
```

```
CREATE TABLE KEYWORDS (
        KEY_ID  INTEGER NOT NULL,
        WORDS   VARCHAR(40) NOT NULL
);
ALTER TABLE KEYWORDS ADD CONSTRAINT PK_KEYWORDS
        PRIMARY KEY  (KEY_ID);


CREATE TABLE APPVERSION_FRAMVERSION(
        VERSION_ID INTEGER NOT NULL,
        FRAMVERSION_ID INTEGER NOT NULL
);
ALTER TABLE APPVERSION_FRAMVERSION ADD CONSTRAINT
        FK_APPVERSION_FRAMVERSION_1 FOREIGN KEY     (VERSION_ID,
        FRAMVERSION_ID) REFERENCES APPLICATION_VERSION (VERSION_ID,
        APP_ID);


CREATE TABLE APPLICATION (
        APP_ID          INTEGER NOT NULL AUTO_INCREMENT ,
        NAME            VARCHAR(30) NOT NULL,
        OWNER           VARCHAR(30) NOT NULL,
        DESCRIPTION  VARCHAR(200) NOT NULL
);
ALTER TABLE APPLICATION ADD CONSTRAINT PK_APPLICATION
        PRIMARY KEY (APP_ID);


CREATE TABLE APPLICATION_VERSION (
        VERSION_ID          INTEGER NOT NULL AUTO_INCREMENT,
        APP_ID              VARCHAR(40)  NOT NULL,
        APP_VERSION         VARCHAR(5) NOT NULL,
        FRAMVERSION_ID   VARCHAR(70) NOT NULL,
        FRAM_VERSION       VARCHAR(5) NOT NULL,
        CREATE_DATE        DATE NOT NULL,
        DESCRIPTION         VARCHAR(200)
);
ALTER TABLE APPLICATION_VERSION
        ADD CONSTRAINT PK_APPLICATION_VERSION
        PRIMARY KEY (VERSION_ID, APP_ID);
```

APÊNDICE D – Manual do Usuário da Ferramenta

“TOFRA”

A FRAMEWORK VERSION CONTROL SUPPORT TOOL

Marília, 20 de Novembro de 2006.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

1. INTRODUÇÃO

1.1. Visão Geral

O controle de versões de *frameworks* é mais complexo, pois, apesar de *framework* ser considerado também um software, há peculiaridades que devem ser consideradas no controle de versões, em conseqüência da necessidade de controlar tanto as versões dos *frameworks*, como as versões das aplicações.

A ferramenta TOFRA foi desenvolvida com o objetivo de apoiar o controle de versões de aplicações baseadas em *frameworks*, utilizando padrões e técnicas existentes, a fim de colaborar para o aumento da garantia de qualidade do software sob essa perspectiva.

A ferramenta trata basicamente do gerenciamento das versões dos *frameworks* e das aplicações vinculadas a cada versão do *framework*. Todas as informações são armazenadas em um repositório para posterior análise. Armazena-se também todos os requisitos (funcionais e não funcionais) cobertos por cada versão do *framework* (requisitos implementados no *framework*), e os requisitos não cobertos pelo *framework*. (requisitos implementados manualmente em alguma versão de uma aplicação). Os requisitos não cobertos poderão futuramente ser implementados no *framework*, seguindo as diretivas e exigências do PREF (CAGNIN, 2005).

Outro ponto importante é que a ferramenta TOFRA também apóia o controle de versões de *framework* orientado a aspectos, em que mais de um *framework* pode ser vinculado a uma determinada aplicação, diferentemente dos *frameworks* convencionais.

2. USUÁRIOS DA FERRAMENTA

Foram definidos quatro tipos de usuários, cada um com suas particularidades:

- **Administrador do Sistema:** o administrador do sistema tem acesso irrestrito a todas as funcionalidades da ferramenta, desde a inserção, alteração e remoção dos dados, independente do usuário que tenha efetuado o cadastro. Além disso, o administrador do sistema é responsável pelo gerenciamento dos diferentes tipos de usuários e pelas permissões de acesso aos interessados na utilização da ferramenta.
- **Proprietário do *Framework*:** o proprietário do *framework* tem permissão de acesso às funcionalidades relacionadas aos *frameworks* e as aplicações vinculadas aos *frameworks*, com exceção das operações de remoção e atualização dos dados, que são exclusivas do administrador do sistema.
- **Proprietário da Aplicação:** o proprietário da aplicação tem acesso as funcionalidades relacionadas com as aplicações vinculadas aos *frameworks*.
- **Usuário Comum:** o usuário comum tem acesso bastante limitado. Seu acesso permite apenas o processamento de consultas e visualização dos dados.

2.1. Autenticação de Usuários

O primeiro passo para a utilização da ferramenta é solicitar uma conta de usuário. Para isso basta clicar no *link* da tela inicial da ferramenta, denominado Formulário de Registro *(Register Form*). O usuário deverá preencher corretamente com os seus dados, inclusive informando qual o seu propósito e, posteriormente, aguardar uma autorização realizada pelo administrador do sistema. O administrador do sistema analisará o propósito de cada um e concederá ou não a permissão solicitada. Na Figura 2.1 é apresentado o formulário de registro.

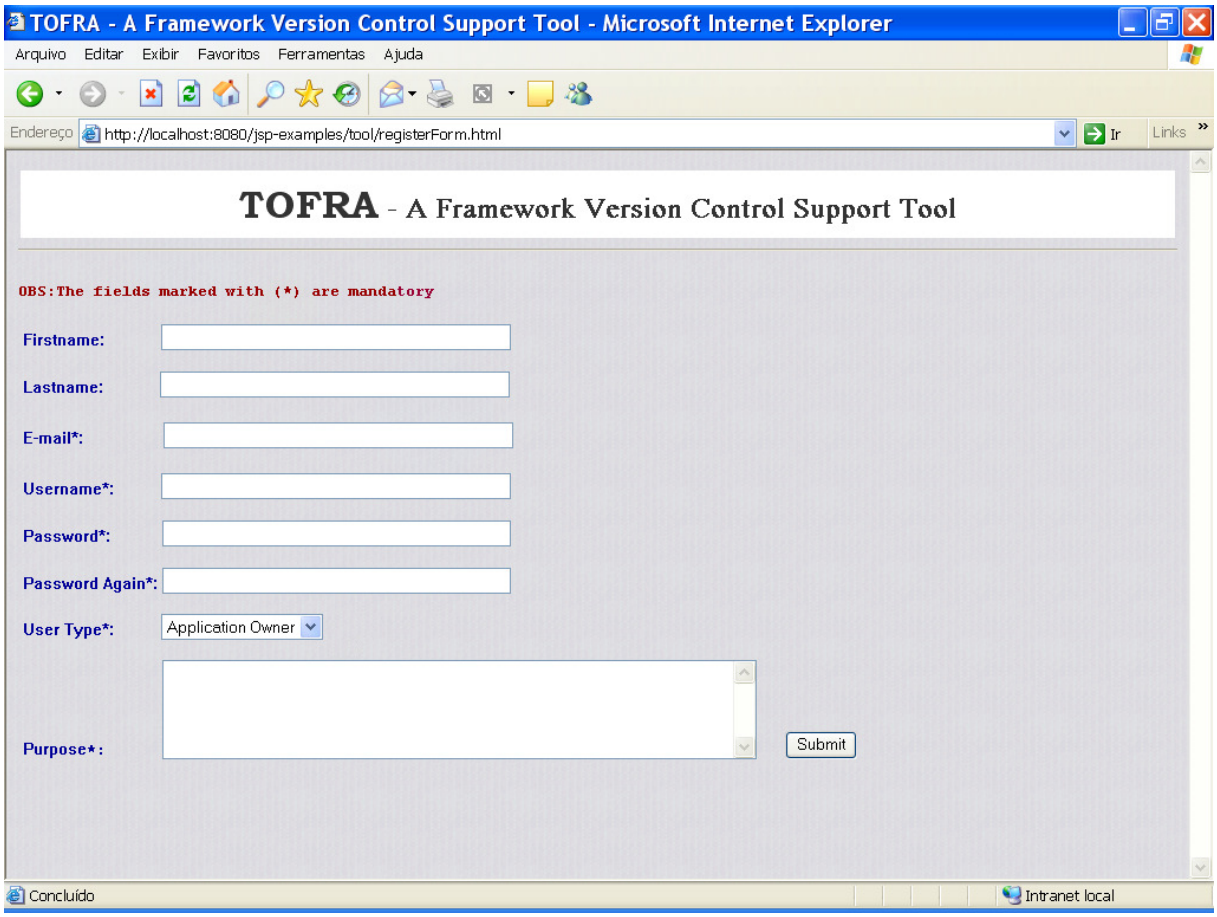


Figura 2.1: Formulário de registro *(Register Form)*

O usuário que já possui uma conta de acesso deverá inicialmente informar seu nome de usuário *(username)* e senha *(password)* na tela de *login* da ferramenta, para que ele possa ser identificado. Efetuado o *login*, o usuário será direcionado a sua página principal, com as funcionalidades correspondentes para cada um.

3. FUNCIONALIDADES DA FERRAMENTA

Nesta seção apresentam-se todas as funcionalidades da ferramenta TOFRA, descritas em detalhes. É importante destacar que qualquer modificação na base de dados é realizada somente pelo administrador do sistema.

3.1. Cadastros

3.1.1. Pré-Usuários (Pré-Users)

Por meio desta funcionalidade é possível alterar, remover e autorizar as pessoas que solicitaram uma conta de usuário para poder utilizar a ferramenta. Entretanto, somente o administrador do sistema tem acesso a esta funcionalidade, que pode ser acessada a partir do link *Users* no menu. Ao clicar nesse *link* uma nova janela será aberta com um registro de todos as pessoas que preencheram o formulário de registro.

- **Autorizar:** para autorizar um pré-usuário a utilizar a ferramenta, selecione o pré-usuário por meio do botão de seleção correspondente e clique no botão *Authorize User.*
- **Alterar:** para alterar os dados de um pré-usuário, selecione o pré-usuário por meio do botão de seleção correspondente e clique no botão *Edit User*. Posteriormente é exibido na tela o formulário, em que poderão ser realizadas as respectivas alterações. Para confirmar a operação, clique em *Save Edit.*
- **Remover:** para remover um pré-usuário, selecione o pré-usuário por meio do botão de seleção correspondente, em seguida basta clicar no botão *Remove Selected* .

3.1.2. Usuários (Users)

Por meio desta funcionalidade é possível inserir, alterar e remover um determinado usuário. Entretanto, somente o administrador do sistema tem acesso a esta funcionalidade, que pode ser acessada a partir do link *Users* no menu. Ao clicar nesse *link* uma nova janela será aberta com um registro de todos os usuários cadastrados na base de dados.

- **Inserir:** para inserir um usuário, clique no botão *Add New User*, em seguida exibe-se na tela um formulário para ser preenchido com os dados do respectivo usuário. Para confirmar a operação, clique no botão *Save User.*
- **Alterar:** para alterar os dados de um usuário, selecione o usuário por meio do botão de seleção correspondente e clique no botão *Edit User*. Posteriormente é exibido na tela o formulário, em que poderão ser realizadas as respectivas alterações. Para confirmar a operação, clique em *Save Edit.*
- **Remover:** para remover um usuário, selecione o usuário por meio do botão de seleção correspondente, em seguida basta clicar no botão *Remove Selected.*

3.1.3. *Frameworks*

Por meio desta funcionalidade é possível inserir, alterar e remover um determinado *framework*. Somente usuários do tipo administrador do sistema e proprietário do *framework* têm acesso a esta funcionalidade, que pode ser acessada a partir do link *Frameworks,* disponível no menu. Ao clicar nesse *link* uma nova janela será aberta com um registro de todos os *frameworks* cadastrados na base de dados

- **Inserir:** para inserir um *framework*, clique no botão *Add New Framework*, em seguida exibe-se na tela um formulário que deve ser preenchido com os dados do

respectivo *framework*. Para confirmar a operação, clique em *Save Framework.* Caso o domínio do *framework* ainda não esteja cadastrado, clique no link *Add New Domain* para cadastrá-lo.

- **Alterar:** para alterar os dados de um *framework*, selecione o *framework* por meio do botão de seleção correspondente e clique no botão *Edit Framework.* Posteriormente é exibido na tela o formulário, em que poderão ser realizadas as respectivas alterações. Para confirmar a operação, clique em *Save Edit.*
- **Remover:** para remover um *framework*, selecione o *framework* por meio do botão de seleção correspondente, em seguida basta clicar no botão *Remove Selected*.

3.1.4. Versões de *Frameworks* (*Frameworks* Versions)

Por meio desta funcionalidade é possível inserir, alterar e remover uma determinada versão de um *framework*. Somente usuários do tipo administrador do sistema e proprietário do *framework* têm acesso a esta funcionalidade, que pode ser acessada a partir do link *Versões de Frameworks* disponível no menu. Ao clicar nesse *link* uma nova janela será aberta com um registro de todos as versões de *frameworks* cadastradas na base de dados.

- **Inserir:** para inserir uma versão de um *framework*, clique no botão *Add New Version,* em seguida exibe-se na tela um formulário que deve ser preenchido com os dados da respectiva versão do *framework*. Para confirmar a operação, clique em *Save Framework Version.*
- **Alterar:** para alterar os dados de uma versão de um *framework*, selecione o *framework* por meio do botão de seleção correspondente e clique no botão *Edit Version.* Posteriormente é exibido na tela o formulário, em que poderão ser

realizadas as respectivas alterações. Para confirmar a operação, clique em *Save Edit.*

- **Remover:** para remover uma versão de um *framework*, selecione a versão por meio do botão de seleção correspondente, em seguida clique no botão *Remove Selected.*

3.1.5. Domínios (Domain Type)

Por meio desta funcionalidade é possível inserir, alterar e remover um determinado domínio de um *framework*. Somente usuários do tipo administrador do sistema e proprietário do *framework* têm acesso a esta funcionalidade, que pode ser acessada a partir do link *Domain Type* disponível no menu. Ao clicar nesse *link* uma nova janela será aberta com um registro de todos os domínios cadastrados na base de dados.

- **Inserir:** para inserir um domínio, clique no botão *Add New Domain,* em seguida exibe-se na tela um formulário que deve ser preenchido com os dados do respectivo domínio. Para confirmar a operação, clique em *Save Domain.*
- **Alterar:** para alterar os dados de um domínio, selecione o domínio por meio do botão de seleção correspondente e clique no botão *Edit Domain.* Posteriormente é exibido na tela o formulário, em que poderão ser realizadas as respectivas alterações. Para confirmar a operação, clique em *Save Edit.*
- **Remover:** Para remover um domínio de um *framework*, selecione o domínio por meio do botão de seleção correspondente, em seguida clique no botão *Remove Selected* .

3.1.6. Requisitos Cobertos pelo *Framework* (Requirement Covered)

Por meio desta funcionalidade é possível inserir, alterar e remover os requisitos (funcionais e não funcionais) implementados/atendidos em cada versão do *framework*. Funcionalidade permitida para usuários do tipo administrador do sistema e proprietário do *framework*, e acessível a partir do link *Requirements Covered* no menu do usuário. Ao clicar nesse *link* uma nova janela será aberta com um registro de todos os requisitos cobertos pelo *framework* cadastrados na base de dados. Na Figura 3.1 é possível visualizar esse registro.

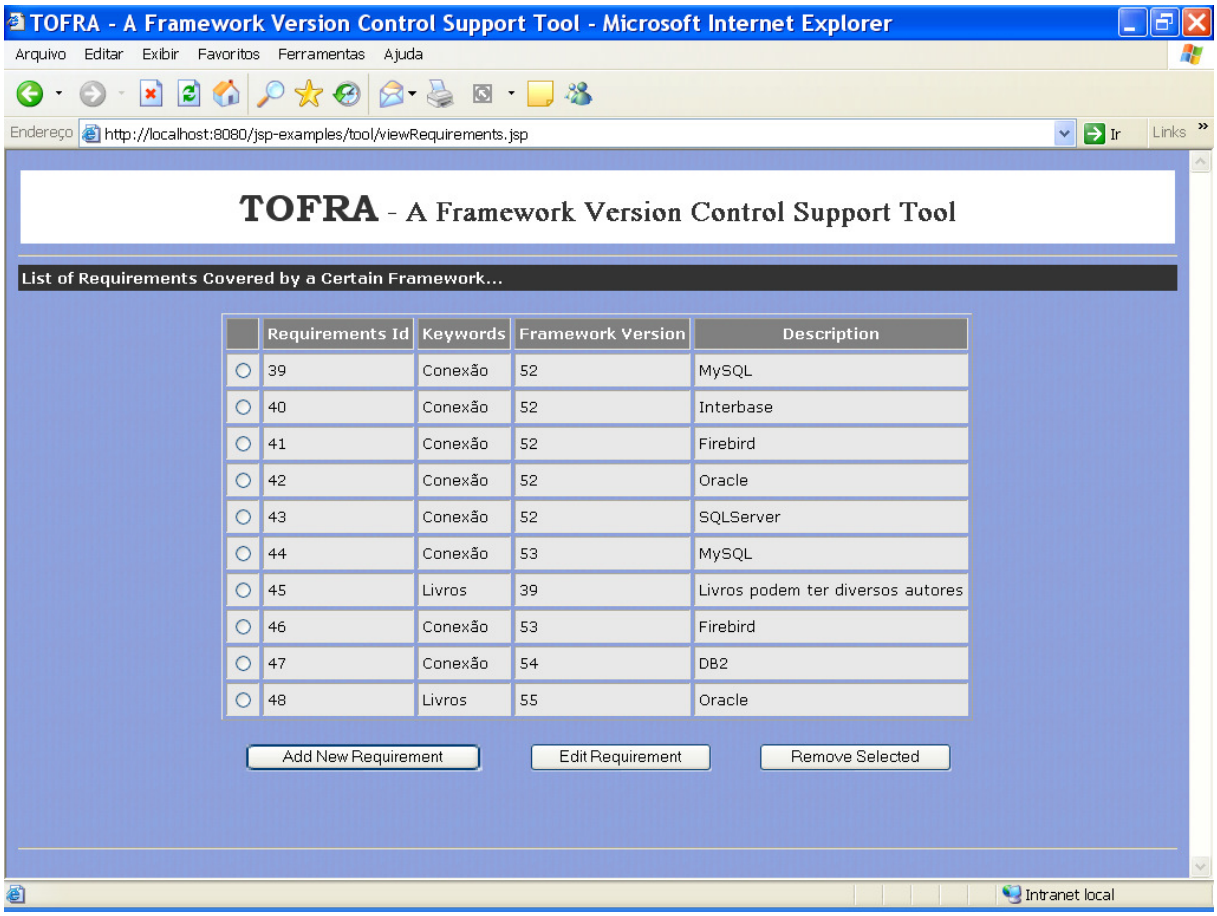


Figura 3.1 – Tela de visualização de requisitos cobertos pelo *framework*

- **Inserir:** para inserir um requisito, clique no botão *Add New Requirement,* em seguida exibe-se na tela um formulário que deve ser preenchido com os dados do

respectivo requisito. Para confirmar a operação, clique em *Save Requirement.* Caso haja a necessidade de cadastrar a palavra-chave do requisito, clique no link *Add New Keywords* para cadastrá-la.

- **Alterar:** para alterar os dados de um requisito, selecione o requisito por meio do botão de seleção correspondente e clique no botão *Edit Selected.* Posteriormente é exibido na tela o formulário, em que poderão ser realizadas as respectivas alterações. Para confirmar a operação, clique em *Save Edit.*
- **Remover:** para remover um requisito, selecione o requisito por meio do botão de seleção correspondente, em seguida basta clicar no botão *Remove Selected.*

3.1.7. Requisitos não Cobertos pelo *Framework* (Requirements not Covered)

Por meio desta funcionalidade é possível inserir, alterar e remover os requisitos (funcionais e não funcionais) não cobertos pelos *frameworks* e que foram implementados manualmente em alguma versão de uma aplicação. Funcionalidade permitida para usuários do tipo administrador do sistema, proprietário do *framework* e proprietário da aplicação, e acessível a partir do link *Requirements not Covered* do menu usuário. Ao clicar nesse *link* uma nova janela será aberta com um registro de todos os requisitos não cobertos pelo *framework* cadastrados na base de dados.

- **Inserir:** para inserir um requisito não coberto pelo *framework*, clique no botão *Add New Requirement,* em seguida exibe-se na tela um formulário que deve ser preenchido com os dados do respectivo requisito. Para confirmar a operação, clique em *Save Requirement.*
- **Alterar:** para alterar os dados de requisito não coberto pelo *framework*, selecione o requisito por meio do botão de seleção correspondente e clique no botão *Edit*

Selected. Posteriormente é exibido na tela o formulário, em que poderão ser realizadas as respectivas alterações. Para confirmar a operação, clique em *Save Edit.*

- **Remover:** para remover um requisito não coberto pelo *framework*, selecione o requisito por meio do botão de seleção correspondente, em seguida basta clicar no botão *Remove Selected*.

3.1.8. Palavras-Chave (Keywords)

Por meio desta funcionalidade é possível inserir, alterar e remover uma palavra-chave que será associada a cada requisito, servindo como parâmetro de consulta. Funcionalidade permitida para usuários do tipo administrador do sistema, proprietário do *framework* e proprietário da aplicação, e acessível a partir do link *Keywords* no menu do usuário. Ao clicar nesse *link* uma nova janela será aberta com um registro de todas as palavras-chaves cadastradas na base de dados.

- **Inserir:** para inserir uma palavra-chave, clique no botão *Add New Keywords,* em seguida exibe-se na tela um formulário que deve ser preenchido com os dados da respectiva palavra-chave. Para confirmar a operação, clique em *Save Keywords.*
- **Alterar:** para alterar os dados de uma palavra-chave, selecione a palavra-chave por meio do botão de seleção correspondente e clique no botão *Edit Selected.* Posteriormente é exibido na tela o formulário, em que poderão ser realizadas as respectivas alterações. Para confirmar a operação, clique em *Save Edit.*
- **Remover:** para remover uma palavra-chave, selecione a palavra-chave por meio do botão de seleção correspondente, em seguida basta clicar no botão *Remove Selected.*

3.1.9. Aplicações (Applications)

Por meio desta funcionalidade é possível inserir, alterar e remover uma determinada aplicação. Funcionalidade permitida para usuários do tipo administrador do sistema, proprietário do *framework* e proprietário da aplicação, disponível no menu do usuário é acessível a partir do link *Applications*. Ao clicar nesse *link* uma nova janela será aberta com um registro de todas as aplicações cadastradas na base de dados.

- **Inserir:** para inserir uma aplicação, clique no botão *Add New Application,* em seguida exibe-se na tela um formulário que deve ser preenchido com os dados da respectiva aplicação (Figura 3.2). Para confirmar a operação, clique em *Save Application.*

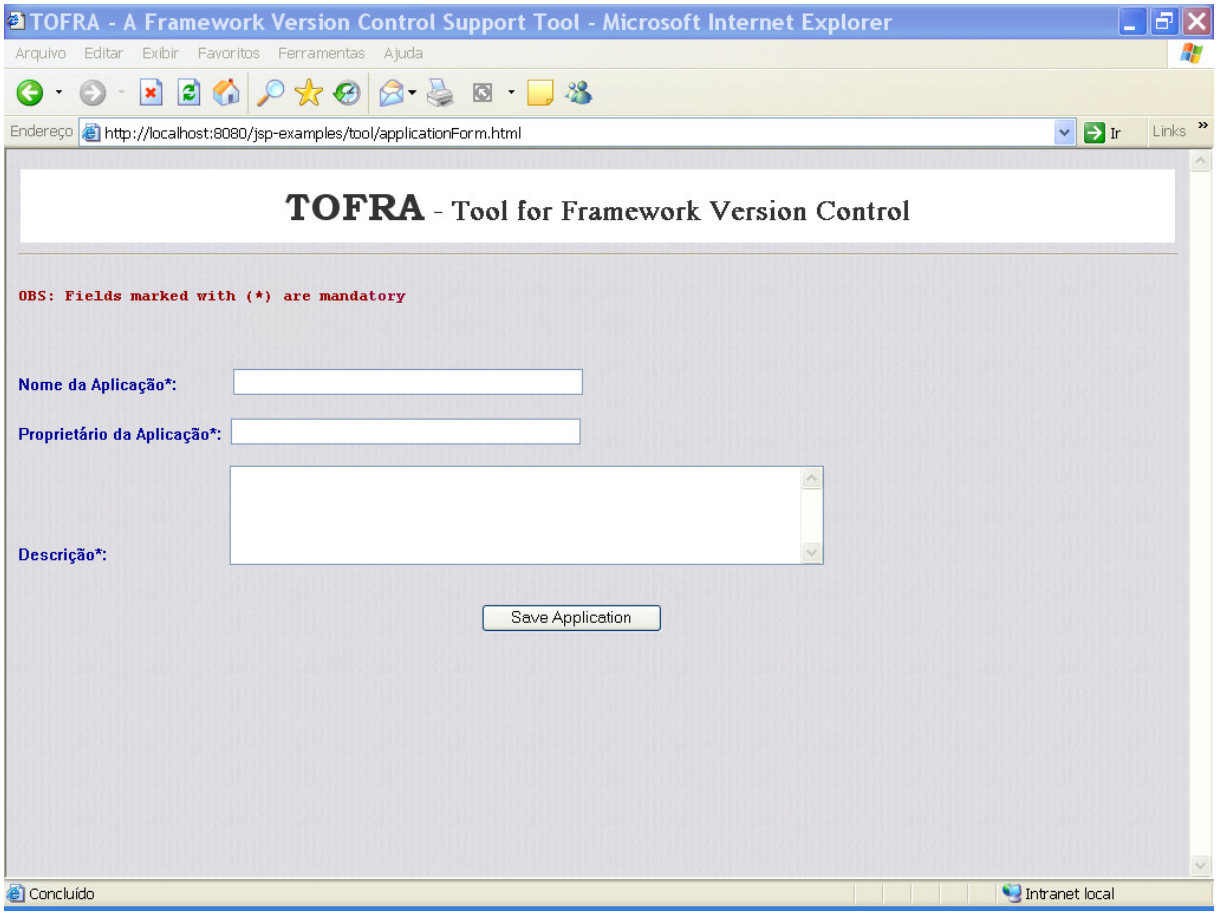


Figura 3.2 – Formulário de cadastro de aplicações

- **Alterar:** para alterar os dados de uma aplicação, selecione a aplicação por meio do botão de seleção correspondente e clique no botão *Edit Application.* Posteriormente é exibido na tela o formulário, em que poderão ser realizadas as respectivas alterações. Para confirmar a operação, clique em *Save Edit.*
- **Remover:** para remover uma aplicação, selecione a aplicação por meio do botão de seleção correspondente, em seguida basta clicar no botão *Remove Selected.*

3.1.10. Versões de Aplicações (Application Versions)

Por meio desta funcionalidade é possível inserir, alterar e remover uma determinada versão de uma aplicação. Funcionalidade permitida para usuários do tipo administrador do sistema, proprietário do *framework* e proprietário da aplicação, disponível no menu do usuário é acessível a partir do link *Application Versions*. Ao clicar nesse *link* uma nova janela será aberta com um registro de todas as versões de aplicações cadastradas na base de dados

- **Inserir:** para inserir uma versão de uma aplicação, clique no botão *Add New Version,* em seguida exibe-se na tela um formulário que deve ser preenchido com os dados da respectiva versão. Para confirmar a operação, clique em *Save Version.*
- **Alterar:** para alterar os dados de uma versão de uma aplicação, selecione a versão da aplicação por meio do botão correspondente e clique no botão *Edit Version.* Posteriormente é exibido na tela o formulário, em que poderão ser realizadas as respectivas alterações. Para confirmar a operação, clique em *Save Edit.*
- **Remover:** para remover uma versão de uma aplicação, selecione a versão por meio do botão de seleção correspondente, em seguida basta clicar no botão *Remove Selected*.

3.2. Consultas

3.2.1. *Frameworks*

Por meio desta funcionalidade é possível consultar os dados de um determinado *framework*. Funcionalidade permitida para todos os tipos de usuários cadastrados, e acessível a partir do link *Frameworks* no menu de consulta do usuário. Para efetuar uma consulta clique no link correspondente, em seguida exibe-se na tela uma nova janela com o formulário de consulta que pode ser visualizado na Figura 3.3. Nesse formulário informe como parâmetro o nome do *framework* e clique no botão *Search* para prosseguir com a pesquisa.

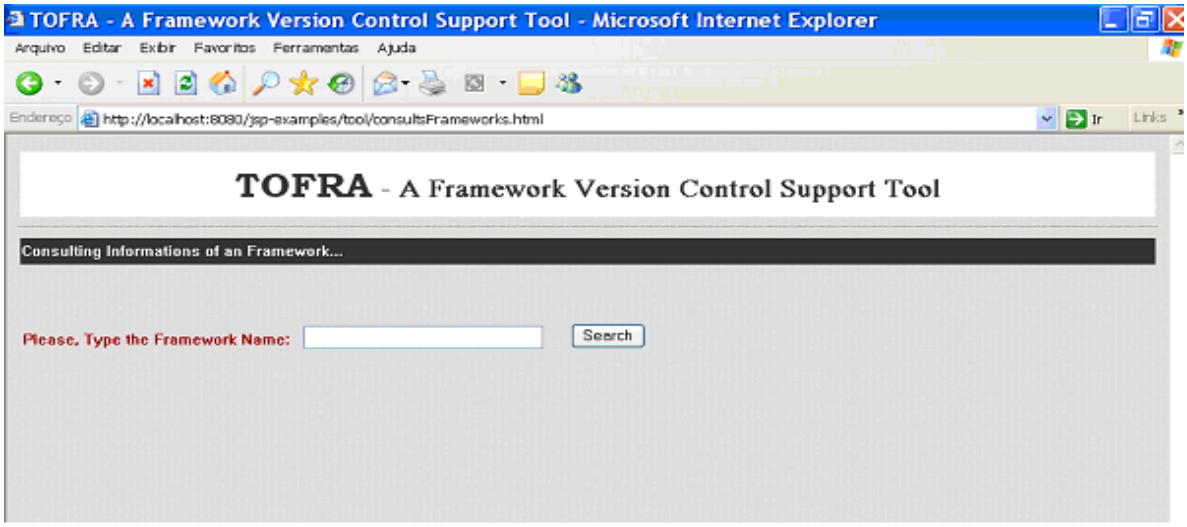


Figura 3.3 – Formulário de consulta de *frameworks*

3.2.2. Versões de *Frameworks* (*Framework* Versions)

Por meio desta funcionalidade é possível consultar os dados de uma determinada versão do *framework*. Funcionalidade permitida para todos os tipos de usuários cadastrados, e acessível a partir do link *Framework Version* no menu de consulta do usuário. Para efetuar a

consulta clique no link correspondente, em seguida exibe-se na tela uma nova janela com o formulário de consulta. Nesse formulário informe como parâmetro o código do *framework* e clique no botão *Search* para prosseguir com a pesquisa.

3.2.3. Requisitos Cobertos por cada Versão do *Framework* (Requirements Covered)

Por meio dessa funcionalidade é possível consultar os requisitos (funcionais e não funcionais) cobertos pelo *framework*, ou seja, implementados/atendidos em cada versão do *framework*. Funcionalidade permitida para todos os tipos de usuários cadastrados, e acessível a partir do link *Frameworks* no menu de consulta. Para efetuar uma consulta clique no link correspondente, em seguida exibe-se na tela uma nova janela com o formulário de consulta. Nesse formulário informe como parâmetro a palavra-chave associada ao requisito e clique no botão *Search* para prosseguir com a pesquisa.

3.2.4. Requisitos não Cobertos pelo *Framework* (Requirements not Covered)

Por meio dessa funcionalidade é possível consultar os requisitos (funcionais e não funcionais) não cobertos pelo *framework* e que foram implementados manualmente em alguma versão de uma aplicação. Funcionalidade permitida para todos os tipos de usuários cadastrados, e acessível a partir do link *Requirements Covered* no menu de consulta. Para efetuar uma consulta clique no link correspondente, em seguida exibe-se na tela uma nova janela com o formulário de consulta. Nesse formulário informe como parâmetro a palavra-chave associada ao requisito e clique no botão *Search* para prosseguir com a pesquisa.

3.2.5. Aplicações (Applications)

Por meio desta funcionalidade é possível consultar os dados de uma determinada aplicação. Funcionalidade permitida para todos os tipos de usuários cadastrados, e acessível a partir do link *Aplicações* no menu de consulta. Para efetuar uma consulta clique no link correspondente, em seguida exibe-se na tela uma nova janela com o formulário de consulta. Nesse formulário informe como parâmetro o nome da aplicação e clique no botão *Search* para prosseguir com a pesquisa.

3.2.6. Versões de Aplicações (Application Versions)

Por meio desta funcionalidade é possível consultar os dados de uma determinada versão de uma aplicação. Funcionalidade permitida para todos os tipos de usuários cadastrados, e acessível a partir do link *Application Versions* no menu de consulta. Para efetuar uma consulta clique no link correspondente, em seguida exibe-se na tela uma nova janela com o formulário de consulta. Nesse formulário informe como parâmetro o código da versão da aplicação e clique no botão *Search* para prosseguir com a pesquisa.

3.2.7. Solução de Projeto (Design Solution)

Por meio desta funcionalidade é possível consultar uma determinada solução de projeto de um requisito que foi implementado manualmente em alguma versão de uma aplicação. Funcionalidade permitida para todos os tipos de usuários cadastrados, e acessível a partir do link *Design Solution* no menu de consulta. Para efetuar uma consulta clique no link correspondente, em seguida exibe-se na tela uma nova janela com o formulário de consulta.

Nesse formulário informe como parâmetro o nome/descrição do requisito e clique no botão *Search* para prosseguir com a pesquisa.