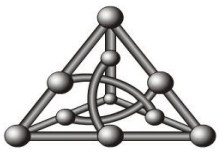
FUNDAÇÃO UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL

ARQUITETURA DO GERADOR DE APLICAÇÃO WEB BASEADO NO FRAMEWORK TITAN

EBERSON OMAR WESCHTER

Dissertação de Mestrado em Ciência da Computação

Área: Engenharia de Software



Departamento de Computação e Estatística

Julho 2008

ARQUITETURA DO GERADOR DE APLICAÇÃO WEB BASEADO NO FRAMEWORK TITAN

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, Curso de Pós-Graduação em Ciência da Computação, Departamento de Computação e Estatística da Fundação Universidade Federal de Mato Grosso do Sul.

Orientador: Prof. Dr. Marcelo Augusto Santos Turine

Julho 2008

Agradecimentos

Começo agradecendo sempre a Deus pelo dom da vida e por me dar saúde.

Ao meu orientador professor Marcelo Augusto Santos Turine pela atenção, apoio, esclarecimentos importantes e confiança durante o decorrer e na conclusão deste trabalho.

A todos os professores e funcionários do DCT, onde fui acolhido sempre que precisei, em especial ao professor Henrique Mongelli, por seu apoio e colaboração como coordenador do Curso de Mestrado em Ciência da Computação do DCT da Universidade Federal de Mato Grosso do Sul.

Aos colegas de mestrado e do LEDES, em especial ao Camilo, pelos preciosos minutos de atenção e contribuição e Ademir, por seu companheirismo e inúmeras horas de estudo em conjunto.

A todos os amigos e colegas, em especial ao Clóvis, pelo apoio, compreensão, descontração e amizade.

Aos meus pais Oldemar e Fani que desde cedo incutiram em mim a importância do estudo e aperfeiçoamento contínuo e meu irmão Edson, pelo grande incentivo e confiança.

Finalmente, um grande e especial agradecimento a Eveline, minha esposa, pelo amor, paciência, compreensão e apoio incondicional em todos os momentos, apresentando sempre uma palavra de incentivo e carinho, e meu pequeno Felipe que, apesar da pouca idade, com sua alegria e vivacidade soube compreender e suportar as horas de ausência.

Sumário

Lista de Figuras e Tabelas

Lista de Abreviaturas e Siglas

ASP - *Active Server Pages*

BNF - *Backus-Naur Form*

CASE - *Computer-Aided Software Engineering*

DBC - Desenvolvimento Baseado em Componentes

DCT – Departamento de Computação e Estatística da UFMS

EJB – *Enterprise Java Beans*

FAP – Fundação de Apoio a Pesquisa

FAST - *Family-Oriented Abstraction, Specification, and Translation*

FUNDECT - Fundação de Apoio ao Desenvolvimento do Ensino, Ciência e Tecnologia do Estado de Mato Grosso do Sul

GUI - *Graphic Users Interface*

HTML - *Hypertext Marckup Language*

J2EE - *Java 2 Platform, Enterprise Edition*

JSP – *Java Server Pages*

LED - Linguagem Específica de Domínio

LEDES - Laboratório de Engenharia de Software do DCT/UFMS

LMA - Linguagem de Modelagem de Aplicações

LPS - Linha de Produtos de Software

MVC - *Model-View-Controller*

PHP - *Personal Home Page*

RI - Representação Intermediária

SAGF - Sistemas *Web* de Apoio à Gestão de Fomento de Projetos

SGBD - Sistema de Gerenciamento de Bancos de Dados

SHA - *Secure Hash Standard*

SIPES - Sistema de Informação em Pesquisa Universitária

SOAP - Simple Object Access Protocol

SQL – *Structured Query Language*

UFMS – Universidade Federal de Mato Grosso do Sul

UML – *Unified Modeling Language*

WIS - *Web Information Systems*

XMI - *XML Metadata Interchange*

XML - *Extensible Marckup Language*

XSL - *Extensible Stylesheet Language*

XSLT - *Extensible Stylesheet Language Transformations*

Arquitetura do Gerador de aplicação Web Baseado no Framework TITAN

Resumo

O crescente aumento do número de aplicações baseadas na Web (*WebApps*) motiva a pesquisa e o desenvolvimento de ferramentas de reuso de artefatos de software já produzidos a fim de auxiliar na geração automática das aplicações. No intuito de padronizar e simplificar esta atividade, a utilização de padrões de software, componentes, *frameworks* e linhas de produtos de software (LPS) são estratégias para tornar mais ágil o desenvolvimento de aplicações, aumentando a reusabilidade dos modelos e do código produzido. Uma forma de automatizar parte do processo de desenvolvimento de software é utilizar geradores de aplicações, possibilitando gerar *WebApps* a partir de especificações em alto nível. Neste contexto, a utilização de ferramentas e técnicas de geração automática de aplicações é o escopo do presente trabalho que objetiva propor um modelo de arquitetura para um gerador automático de *WebApps* integrado à ferramenta Fênix e baseado no *framework* Titan, automatizando o processo de geração de código de *WebApps* no contexto de uma LPS orientada a família de produtos no domínio de Sistemas Web de Apoio à Gestão de Fomento de Projetos (SAGF). O gerador de aplicação é composto por um gerador de código baseado em regras, metadados e transformações aplicadas no *framework* Titan, além de arquivos de configuração da interface da aplicação gerada. Por fim, a ferramenta Fênix permitirá gerar código de uma aplicação a fim de gerenciar o processo de envio, avaliação, monitoramento e finalização de propostas eletrônicas de projeto a serem avaliadas por agências de fomento. Para validar e testar o gerador de aplicação, será gerado código em PHP e banco de dados PostgreSQL para uma aplicação de gerenciamento de projetos de iniciação científica da Pró-Reitoria de Pesquisa e Pós-Graduação da UFMS.

Palavras-chave: Geração de código, gerador de aplicação, reusabilidade, *framework* Titan, linha de produtos de software e ferramenta Fênix.

Capítulo 1
Introdução

1.1. Considerações Iniciais

O avanço da Internet tem motivado o crescimento de *WebApps* que combinam navegação e interatividade em um grande espaço de documentos heterogêneos. A *Web* tem se mostrado como um dos mais efetivos e atrativos meios de divulgação, de negociação e de entrega de bens e serviços (BALASUBRAMANIAN et al., 1997; BIEBER et al., 1998; CONALLEN, 2000; ROSSI et al., 2001; GLOVER et al., 2002, BRAMBILLA et al., 2006; DISTANTE et al., 2007).

Tendo em vista que *WebApp* é considerada um tipo de produto de software, métodos específicos de Engenharia de Software para a Web têm sido propostos e sistematizam as fases de análise, de projeto e de implementação. Para auxiliar o processo de desenvolvimento de software, os engenheiros utilizam recursos e artefatos (modelos, processos, técnicas, linguagens de programação e ferramentas) fundamentados em um paradigma de desenvolvimento (estruturado, orientado a objetos ou baseado em componentes, por exemplo).

Uma das áreas de pesquisa da Engenharia de Software que está contribuindo para acelerar o processo de desenvolvimento de software é a reutilização, que tem sido referenciada como uma das formas de reuso de artefatos de software para aumentar a produtividade, melhorar a confiança do produto e diminuir custos (FRAKES, KANG, 2005, DENGER, KOLB, 2006). O desenvolvimento de software baseado em reuso induz a um ciclo de vida no qual se distinguem duas grandes atividades: elaboração de soluções genéricas e reuso destas soluções para o desenvolvimento de softwares específicos.

Segundo Tracz (1995) e reforçado em (BARRY, 1999; WEBER, NASCIMENTO, 2002; OLIVEIRA, BACILLI, 2006), a produtividade dos projetos

aumenta em torno de 50% com o uso de técnicas de reutilização de software. Assim, surgem várias iniciativas para a reutilização de soluções para problemas recorrentes, dentre elas, classes abstratas (PREE, 1995), *frameworks* (FAYAD, 1997; BOSCH, 2001; BHATTI et al., 2005), padrões de projeto (GAMMA et al., 1995; COALLIER, 2007), padrões arquiteturais (BUSCHMANN et al., 1996; BASS et al., 2002; GOAER et al., 2008), componentes (SAMETINGER, 1997; LAU, 2006), geradores de aplicação (SMARAGDAKIS, BATORY, 2002; BATORY, 2005; WANG et al., 2008) e linha de produtos de software (LPS) (CLEMENTS, NORTHROP, 2002; BATORY et al., 2002; GILL, 2006).

Diante deste panorama e das tendências tecnológicas, várias pesquisas estão sendo realizadas objetivando propor modelos, técnicas, métodos e ferramentas automatizadas para auxiliar o processo de desenvolvimento de *WebApps*. Surge assim, uma oportunidade para especificar e implementar novas técnicas e ferramentas utilizando LPS e geradores de aplicação: contexto deste trabalho.

1.2. Motivações

Uma das pesquisas em evolução na área de reuso de software é baseada em geradores de aplicação, que automatizam a implementação de artefatos de software, tais como, código fonte e respectiva documentação, constituindo-se em uma importante técnica de reuso de software. É de consenso na literatura (BOSCH et al., 1999; CRNKOVIC et al., 2002; GIMENES, TRAVASSOS, 2002; BRAGA, 2003; CZARNECKI, 2004; BALZERANI et al., 2005, BATORY, 2005) que a partir do reuso de modelos, de projeto e de partes da implementação de software, pode-se construir novos produtos em menor tempo e com maior confiabilidade.

Neste contexto, o grupo de pesquisa em Engenharia de Software do Departamento de Computação e Estatística (DCT) da UFMS, por meio do Laboratório de Engenharia de Software (LEDES), está desenvolvendo vários trabalhos de pesquisa na área de reuso de *WebApps* (ARAGON, 2004; ISHY, 2004; TURINE, ISHY, 2004; TURINE et al. 2005a; TURINE et al. 2005b; CARROMEU, TURINE, 2005; CARROMEU, TURINE, 2006a; CARROMEU, TURINE, 2006b; CARROMEU, 2007), o que motiva a presente pesquisa.

Este projeto é uma continuidade do trabalho de mestrado defendido em 2007 sobre a ferramenta Fênix (CARROMEU, 2007; CARROMEU, TURINE, 2006a, CARROMEU; TURINE, 2006b), um projeto financiado pela FUNDECT (Fundação de Apoio ao Desenvolvimento do Ensino, Ciência e Tecnologia do Estado de Mato Grosso do Sul) (http://www.fundect.ms.gov.br), cujo objetivo é instanciar e gerar *WebApps* a fim de auxiliar a gestão (submissão, avaliação, monitoramento e finalização) de propostas eletrônicas de projeto a serem avaliadas por agências de fomento. Na arquitetura da ferramenta Fênix, um dos subsistemas é o gerador de aplicação, objeto da presente pesquisa.

Atualmente, existem 28 instituições de fomento (FAPs) à pesquisa no Brasil, e a maioria ainda utiliza meios não digitais de gerenciamento e acompanhamento de projetos, e poucas possuem sistemas Web dinâmicos para interação com sua comunidade. Neste domínio não existem softwares genéricos e tampouco padrões pré-definidos para fabricação/desenvolvimento eficaz de produtos Web.

Desta forma, torna-se necessário desenvolver sistemas informatizados e genéricos de gestão e de acompanhamento de propostas de projetos de pesquisa, de forma a tornar todo o processo de fomento eficiente e eficaz. Além disso, tais sistemas devem possuir a propriedade de assegurar um registro fiel da produção científica regional e garantir, de forma transparente, o acompanhamento de investimentos de capital nesta área. Esta problemática motiva a busca por soluções para aperfeiçoar este processo nas FAPs.

1.3. Objetivos

O objetivo geral deste trabalho é propor uma arquitetura para o gerador automático de *WebApps* baseado em *framework* e integrado à ferramenta Fênix, que automatiza o processo de desenvolvimento de LPS orientado a família de produtos no domínio de Sistemas *Web* de Apoio à Gestão de Fomento de Projetos (SAGF), permitindo a geração de código e de estilos de interface para a aplicação gerada.

Os objetivos específicos da pesquisa são:

- Revisar a literatura sobre modelos, ferramentas e principais contribuições teóricas existentes sobre geradores de aplicações e geradores de código;
- Estabelecer os passos necessários para modelagem e geração de aplicação no domínio de apoio a gestão de fomentos, utilizando, como infra-estrutura tecnológica básica a arquitetura da ferramenta Fênix;
- Propor um modelo de arquitetura para implementação de geradores de *WebApp* no domínio de gestão de fomentos;
- Especificar e implementar o módulo gerador de código na ferramenta Fênix utilizando como base a estrutura do *framework* subjacente à ferramenta Fênix,
- Especificar e implementar um *wizard* para facilitar a geração de código; e
- Validar e testar o gerador de aplicação utilizando um estudo de caso no domínio de avaliação de projetos de iniciação científica da Pró-Reitoria de Pesquisa e Pós-Graduação da UFMS.

1.4. Organização do Texto

Este documento está estruturado da seguinte forma: no Capítulo 2 é apresentada uma revisão sobre reusabilidade de software, destacando temas como desenvolvimento baseado em componentes, *frameworks*, linha de produtos de software e comparação entre geradores de código e de aplicação, além de destacar trabalhos relacionados ao tema da presente proposta. No Capítulo 3 é apresentado o projeto Fênix em desenvolvimento no DCT/UFMS, destacando sua arquitetura e o *framework* Titan (LEDES, 2006) subjacente. No Capítulo 4 é apresentada a proposta da arquitetura do gerador de aplicações do Fênix e tecnologias adotadas. O Capítulo 5 mostra o processo de geração e instanciação de uma aplicação final e, finalmente, no Capítulo 6 são apresentadas as considerações finais e sugestões de trabalhos futuros.

Capítulo 2
Reusabilidade

2.1. Considerações Iniciais

A reutilização de software é uma das áreas da Engenharia de Software que propõe um conjunto sistemático de processos, técnicas e ferramentas para melhorar a produtividade, a manutenibilidade e a qualidade tanto do software quanto do processo do desenvolvimento. Caracteriza-se pela utilização de artefatos de software em uma situação diferente da qual foram originalmente projetados e construídos. É de consenso na literatura (BENGTSSON et al., 1999; SILVA, 2000; GIMENES; TRAVASSOS, 2002; BRAGA, 2003; BATORY, 2005; WANG et al., 2008, GOAER et al., 2008) que a partir do reuso de modelos, de projeto e de partes da implementação de software, pode-se construir novos produtos em menor tempo e com maior confiabilidade.

Neste contexto, diversas técnicas e plataformas de reuso têm sido propostas na literatura, tais como, *frameworks*, desenvolvimento baseado em componentes (DBC), LPS e geradores de aplicação. Neste capítulo é apresentada uma breve revisão dessas técnicas que enfatizam e/ou propiciam o reuso: *frameworks* e DBC são apresentados na Seção 2.2; na Seção 2.3 é tratado o reuso por meio de LPS; na Seção 2.4 são enfatizados os geradores de aplicação e de código; e na Seção 2.5 são mencionados exemplos de geradores de aplicação e outros trabalhos relacionados na temática deste trabalho; e por fim, na Seção 2.6 as considerações finais do capítulo.

2.2. Componentes e *Frameworks*

A forma mais elementar de reutilização de artefatos de software é o reuso de linhas de código, como, por exemplo, o uso de bibliotecas de funções na

linguagem C. Porém, diferentes abordagens mais estruturadas buscam melhorar a qualidade dos artefatos de software por meio de reusabilidade, dentre as quais se destacam os *frameworks* e componentes de software.

Frameworks podem ser definidos como um esqueleto da aplicação que será instanciada por um desenvolvedor (PREE, SIKORA, 1997; BHATTI et al., 2005). Trata-se de uma aplicação semicompleta reutilizável que, quando especializada, produz aplicações personalizadas (FOOTE, JOHNSON, 1988). Pode ser definido como conjunto de blocos de software pré-fabricados que desenvolvedores podem usar, estender e adaptar para soluções computacionais específicas (FAYAD, 1997). *Frameworks* cumprem com a função principal das técnicas de reutilização "fazer com que desenvolvedores não precisem começar do nada para especificar e construir uma nova aplicação".

A estrutura de um *framework* é formada por um conjunto de classes que contém o projeto abstrato de soluções para uma família de problemas, propiciando o reuso com granularidade maior do que as classes (FOOTE; JOHNSON, 1988). É composto por uma coleção de classes abstratas e concretas e de interfaces entre elas, representando o projeto de um subsistema (SOMMERVILLE, 2006).

O Desenvolvimento Baseado em Componentes (DBC) é uma alternativa para a obtenção de soluções reutilizáveis. Tais necessidades, aliadas às mudanças que estão ocorrendo na computação (por exemplo, computação distribuída, desenvolvimento *Web*, necessidade de estabelecimento de padrões para a construção de aplicações), são os principais fatores que motivam o uso de DBC.

Segundo Sametinger (1997), "componentes de software reutilizáveis são artefatos autocontidos, claramente identificáveis, que descrevem ou realizam uma função específica e têm interfaces claras, documentação apropriada e um grau de reutilização definido". Um componente autocontido não precisa de outro componente para ser reutilizado. Quando existe a necessidade de cooperação entre componentes, esta interação deve ser feita por meio de suas interfaces, ou seja, conjuntos de assinaturas de operações existentes no componente que podem ser invocadas por outro.

Para viabilizar a reutilização de um componente, este deve ser identificável, ou seja, deve ser facilmente localizado e bem documentado, sendo possível avaliar o seu potencial de reutilização por meio das funcionalidades descritas (por exemplo, descrição das interfaces e casos de uso).

Gimenes e Huzita (2005) e Lau (2006) identificam vários benefícios relacionados à reutilização, especificamente de componentes, tais como: redução de custo e tempo de desenvolvimento; gerenciamento da complexidade; desenvolvimento paralelo de componentes que possuem serviços independentes e bem definidos; aumento da qualidade, partindo do pressuposto que estes componentes sejam cuidadosamente testados para promover a sua reutilização, e facilidade de manutenção e atualização, em função da identificação clara dos serviços prestados pelos mesmos.

Contudo os autores também destacam que, para alcançar tais benefícios, é necessário tratar a complexidade envolvida nas tarefas de desenvolvimento de componentes "para" e "com" a reutilização, tanto nos seus aspectos técnicos quanto gerenciais. Para Vitharana et al. (2003) e reforçado em Gill (2006), no início do ciclo de vida de um software baseado em componentes, deve-se identificar as estratégias do negócio, estabelecendo os objetivos gerenciais a serem posteriormente traduzidos em características técnicas de componentes.

Observa-se em comum nas abordagens baseadas em *frameworks* e em componentes a possibilidade de reutilização de artefatos de software de alta granularidade, e a caracterização de reuso de projeto e código. Tais características são altamente desejáveis em projetos de implementação de geradores de aplicações.

2.3. Linha de Produtos de Software

O termo Linha de Produtos de Software (LPS) (*Software Product-lines*) é uma referência às linhas de produção das indústrias de manufatura, as quais, no final do século XIX, introduziram uma revolução no processo produtivo, sugerindo o desenvolvimento seqüencial de produtos, baseado em tarefas repetitivas, executadas sempre pelas mesmas pessoas que dispunham dos recursos materiais que necessitavam.

Conforme Cohen (2002) e Clements e Northrop (2002) uma definição de LPS aceita na literatura é "*um conjunto de sistemas que usam software intensivamente, compartilhando um conjunto de características comuns e gerenciadas, que satisfazem as necessidades de um segmento particular de mercado ou missão, e que são desenvolvidos a partir de um conjunto comum de ativos principais e de uma forma preestabelecida*".

A partir desta definição, alguns termos devem ser destacados, pois representam as características mais relevantes de uma linha de produtos: conjunto comum de ativos, forma preestabelecida e segmento particular de mercado ou missão (negócio da organização). O conjunto comum de ativos é a essência da LPS e corresponde a um conjunto de elementos customizáveis, utilizados na construção dos softwares, ou seja, da família de produtos. Entre os ativos estão, por exemplo, componentes de software, modelos de documentos utilizados no processo (artefatos), *design-patterns* utilizados pela equipe de desenvolvimento, documentação dos requisitos comuns à família de produtos, a arquitetura da linha de produtos (que será a base da arquitetura de cada produto gerado) e cronogramas gerenciais.

A forma preestabelecida define o processo de produção de softwares por meio de uma LPS, utilizando planos de produção definidos para cada software que será produzido pela linha. Ao definir um plano de produção, que dará origem a um novo produto, deve-se relacionar quais ativos farão parte deste produto e assim estabelecer um vínculo aos processos de cada ativo utilizado.

Já no segmento particular ou missão, muitas vezes denominado domínio do negócio, refere-se ao corpo de conhecimento ou à área de especialização em que a linha atuará. O domínio está diretamente relacionado a um conjunto de funcionalidades correlacionadas aos produtos que a linha pretende atender. Como a flexibilidade dos ativos (especialmente da arquitetura da linha) é limitada, o modelo exige uma delimitação de um segmento de atuação. Sem essa delimitação, o escopo da LPS poderia ser muito abrangente o que tornaria muito custoso criar e manter o conjunto comum de ativos.

Assim, para se conceber uma LPS é necessário definir conceitos que generalizam aplicações do domínio e que permitam instanciações para aplicações

específicas. Neste sentido, os *frameworks* podem ser integrados à LPS para projetar arquiteturas de software semidefinidas que consistem de um conjunto de unidades individuais e de interconexões entre elas a fim de criar uma infra-estrutura de apoio pré-fabricada para o desenvolvimento de aplicações em um ou mais domínios específicos. A integração de *frameworks* e geradores de aplicação é a solução proposta no projeto Fênix.

2.4. Geradores de Código e de Aplicação

2.4.1. Geradores de Código

Um gerador de código (FERTALJ et al., 2002; WANG et al., 2008) é uma ferramenta de software que permite a partir de diferentes artefatos de entrada gerar o código de uma aplicação em uma linguagem fonte. Este código pode servir como base para o desenvolvimento de uma aplicação ou simplesmente se tornar protótipo. O objetivo é projetar uma base para o desenvolvimento, excluindo a participação trabalhosa dos engenheiros de software em aspectos repetitivos do desenvolvimento de uma aplicação.

Luker (1986) define gerador como um item de software que produz um programa em alguma linguagem-alvo a fim de atender a um conjunto específico de requisitos. Segundo Lim (1998), um gerador é um construtor automático de alto nível que esconde a interconexão manual entre os componentes usando uma linguagem orientada a problema, *template* ou filtro de opções ou ambiente de programação visual. Um gerador possibilita a especificação concisa da aplicação desejada (ou parte dela), e então gera código apropriado e chamadas a procedimentos em alguma linguagem.

Por exemplo, um *wizard* é um software gráfico que recebe uma especificação em alto nível de abstração e a transforma em artefatos de software. Algumas ferramentas CASE geram uma parte do código de uma aplicação, utilizando como entrada diagramas e especificações. Os compiladores são geradores de código, uma vez que recebem uma informação em certo nível de abstração, tais como, a linguagem Java ou C++, e a transformam em uma

linguagem de mais baixo nível, tais como código objeto, *bytecode* ou código de máquina (CZARNECKI, EISENERCKER, 2002).

Apesar do código produzido ser correto, Moreira e Mrack (2003) citam várias características indesejáveis dos geradores de código:

- Funcionamento em uma via: muitas vezes o código é gerado em uma única direção, ou seja, uma vez que o modelo de dados é modificado, a geração deve ser re-executada. Caso o código existente tenha sido modificado pelo programador, tais alterações são perdidas e devem ser refeitas;
- O padrão de código é dependente da ferramenta: o código produzido geralmente não está de acordo com o padrão de codificação da equipe de desenvolvimento, e sim de acordo com as regras definidas no gerador; e
- Código de baixa qualidade: o código gerado não leva em consideração questões de desempenho, otimização, estrutura, integração com outros sistemas ou documentação.

2.4.2. Geradores de Aplicação

A automação da produção de artefatos de software é uma atividade que pode diminuir o tempo e o custo de desenvolvimento das aplicações. Existem diversas técnicas que podem ser utilizadas para a geração de software. Essas técnicas têm o objetivo de fornecer meios para o engenheiro especificar uma aplicação em uma linguagem abstrata de alto nível e, a partir dessa especificação, uma ferramenta realiza a construção automática de artefatos, tais como código-fonte, casos de teste e documentação.

Dentre as técnicas de automação disponíveis na literatura, estão as Linguagens Específicas de Domínio (LEDs), a arquitetura baseada em modelos e os geradores de aplicação. Dessa forma, geradores de aplicação são as aplicações de software necessárias à efetivação das Linguagens Específicas de Domínio (LEDs) como ferramentas de reuso e geração de aplicações sob o foco de Linhas de Produtos de Software e, cuja abordagem se torna indispensável nesta pesquisa.

Nesta seção são apresentadas as características, arquitetura e técnicas para construção de geradores de aplicação, além de uma descrição do relacionamento que estes têm com as LEDs.

2.4.2.1. Definição e Características

Na literatura, o termo gerador de aplicação pode assumir diferentes significados, tais como, compiladores, pré-processadores, meta-funções que geram classes e procedimentos, *wizards* e geradores de código (CLEAVELAND, 1988; SMARAGDAKIS, BATORY, 2000; BATORY, 2005; WANG et al. 2008). Os geradores de aplicação são ferramentas que possuem o fluxo de execução baseado em uma linguagem de alto nível (baseada em padrões ou não), conduzindo o engenheiro de aplicação em uma seqüência de etapas pré-definidas.

É de consenso na literatura que as ferramentas que permitem especificar o produto em um nível mais alto de abstração e objetivam gerar automaticamente um componente ou aplicação em alguma linguagem de programação são denominadas geradores de aplicação. Automatizam parte de um processo rotineiro da atividade de desenvolvimento de software, acelerando o processo de implementação, transformando especificações de alto nível em artefatos, tais como, código fonte, interfaces, documentação, descrição de processos, métricas, estimativas, especificações de projeto e arquitetura. Dessa forma, ao comparar os dois tipos de geradores, pode-se afirmar que os geradores de aplicação englobam os geradores de código, conforme Figura 2.1.

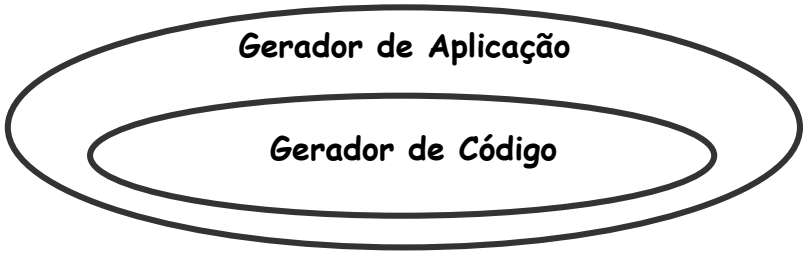


Figura 2.1. Gerador de aplicação e gerador de código.

O uso de geradores de aplicação é justificado em uma organização quando o reuso de um determinado produto de software é realizado em grande escala

(PFLEEGER[1], 1998 *apud* FRANCA, 2000). Isto é, quando existe a demanda de uma grande quantidade de softwares para um mesmo domínio (família de software).

Os principais benefícios no uso de geradores são (CLEAVELAND, 2001; FRANCA, STAA, 2001; HERRINGTON, 2003; FRAKES, KANG, 2005):

- aumento de produtividade: como a maior parte do produto gerado corresponde a trechos fixos referentes a detalhes de implementação tem-se a elevação da produtividade da equipe de desenvolvimento;
- prototipação: novos produtos são gerados alternando-se as especificações fornecidas ao gerador, facilitando assim, o desenvolvimento de novas versões e possibilitando ao desenvolvedor conduzir diversos experimentos para determinar o produto mais adequado ao usuário; e
- qualidade no produto gerado: uma vez que o funcionamento do gerador esteja correto, o produto gerado também estará, sendo que possíveis problemas no produto estão associados a problemas na especificação fornecida ao gerador, que passa constantemente por validação e manutenção.

Segundo (BRAGA, 2003; CZARNECKI, 2004; WANG et al., 2008), os geradores de aplicação constituem uma moderna prática de reuso e, além de reduzir os custos e aumentar a produtividade, ajudam a melhorar a qualidade dos sistemas. O reuso baseado em geradores de aplicação somente é possível quando as abstrações de domínio e seu mapeamento em código executável podem ser definidos, resultando no programa gerado.

Para se construir um gerador de aplicação deve-se entender os requisitos comuns (similaridades) e as variáveis (variabilidades) do domínio do problema para ser possível projetar uma ferramenta de geração eficiente. A necessidade de se modelar as similaridades e as variabilidades do sistema surge naturalmente no decorrer da evolução do processo de desenvolvimento, e torna-se muito importante no reuso (JARZABECK, 1995; HERRINGTON, 2003; OLIVEIRA, BACILLI, 2006).

[1] PFLEEGER, S. L. *Software Engineering Theory and Practice*. Prentice-Hall, NJ, 1998.

2.4.2.2. Geradores de Aplicação e LEDs

As linguagens específicas de domínio (LEDs) constituem uma das formas de especificação para os geradores de aplicação. São linguagens com alto nível de abstração que possuem características específicas para um domínio, permitindo aos desenvolvedores especificarem de forma mais conveniente. Além disso, essas linguagens são muito mais concisas do que as de propósito geral.

As LEDs permitem aos desenvolvedores especificar "o que" a aplicação deve fazer, deixando para o gerador de aplicação a tarefa "de como" fazer.

A visão do mapeamento entre espaço do problema e espaço da solução, ilustrada na Figura 2.2a, também conhecida como modelo de domínio gerativo (CZARNECKI, EISENECKER, 2002, CZARNECKI; 2004), um conceito chave no desenvolvimento gerativo de software, dá uma visão do relacionamento entre LEDs e geradores de aplicações. O espaço do problema é um conjunto de abstrações específicas de domínio que podem ser usadas para especificar um determinado membro de uma família de sistemas. O espaço da solução consiste de abstrações orientadas a implementação, as quais podem ser instanciadas para criar implementações das especificações escritas usando as abstrações específicas do domínio do espaço do problema. O espaço do problema contém uma estrutura que permite a expressão de problemas de forma direta e intencional.

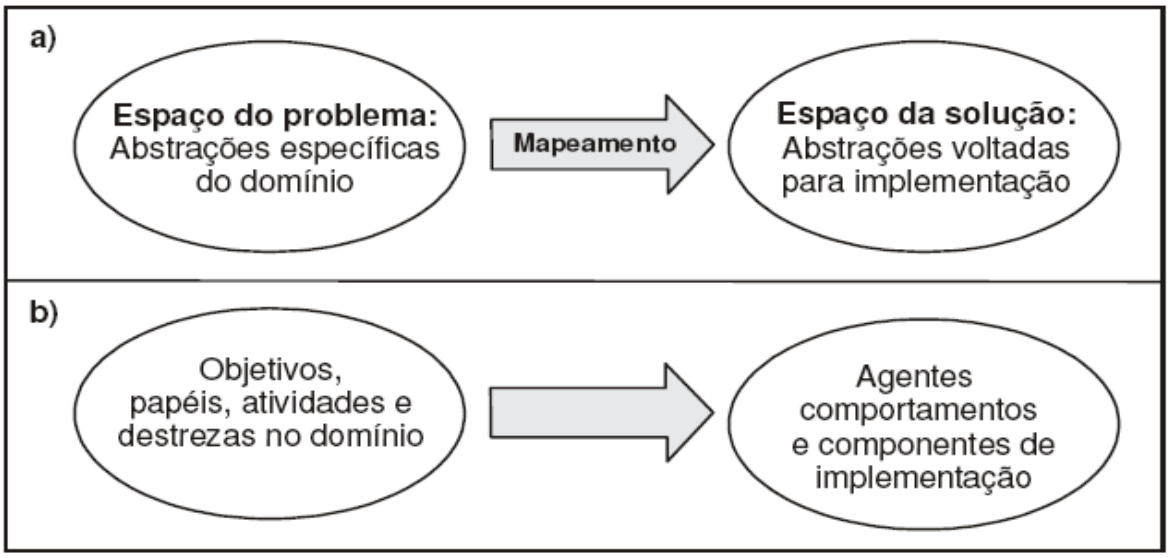


Figura 2.2: Mapeamento entre espaço do problema e espaço da solução.

Nessa visão, a LED está no espaço do problema e o gerador no mapeamento entre os dois espaços, tendo como entrada uma especificação e, como saída, a correspondente implementação. A Figura 2.2b mostra os conceitos presentes nesses espaços: Engenharia de Domínio Específico.

Assim, a LED é a especificação de entrada para o gerador; o gerador é o "processador" da LED, conforme ilustrado na Figura 2.3. A forma textual não é a única existente para a especificação do domínio para o gerador. Esta também pode ser feita por meio de uma GUI (*Graphic Users Interface*), ou interface gráfica de usuário; pode ser uma interface interativa por menu ou combinação das mesmas. Portanto, pode-se denominar todas essas formas de especificação de linguagens específicas de domínio. A Figura 2.3 ilustra o processo de interação entre a LED e o gerador (DELTA SOFTWARE, 2005): o usuário (desenvolvedor) irá traduzir os conceitos do domínio em uma especificação (que pode ser numa das formas descritas anteriormente), que será submetida ao gerador. O gerador então irá automatizar o processo de implementação, traduzindo a especificação para um programa na linguagem-alvo, que será aplicado ao compilador específico da mesma, gerando finalmente a aplicação executável.

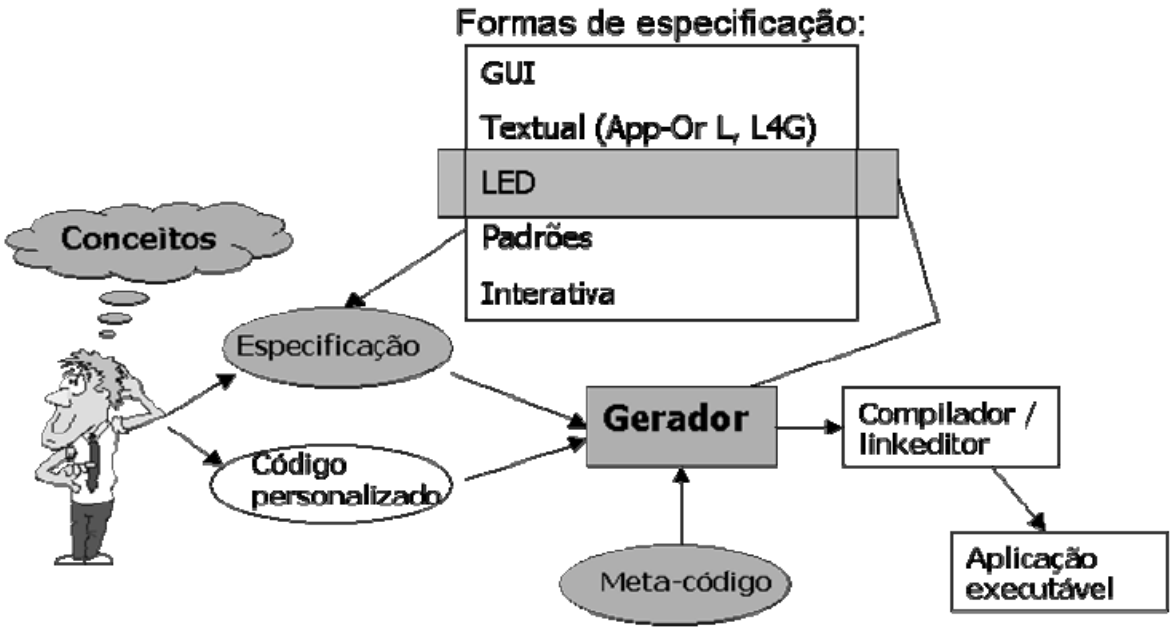


Figura 2.3: Gerador de aplicação e sua relação com uma LED (adaptado de DELTA SOFTWARE, 2007).

A linha que relaciona a LED ao Gerador na Figura 2.3 ilustra que uma LED, para ter sua aplicabilidade completa, precisa de um gerador, mas existem LEDs que nem sempre visam execução, podendo ser usadas somente para especificação, como a BNF (*Backus-Naur Form*).

2.4.2.3. Construção de Geradores de Aplicação

Uma vez que se tem conhecimento e experiência a respeito do desenvolvimento de software em determinado domínio, é desejável que sejam automatizadas as tarefas de desenvolvimento de tais aplicações, reutilizando esse conhecimento e experiência na construção de geradores de aplicação.

Entretanto, para que sejam viáveis a construção e utilização de um gerador de aplicação, certos aspectos têm que ser observados, como: entendimento do domínio do problema; especificação correta, consistente e completa do domínio; domínio bem delimitado e "maduro" ou seja, não esteja sujeito a alterações; necessidade de conhecimento a respeito de linguagens de programação e tecnologia de compiladores.

Segundo Herrington (2003) e Czarnecki (2004), existem duas técnicas principais usadas durante a geração: a composição e a transformação. A composição ou modelo composicional consiste em agrupar um determinado número de componentes que, para maior efetividade, devem ser componentes gerativos em vez de componentes concretos. Por exemplo, para desenhar uma “estrela” precisaríamos de dois componentes gerativos: um círculo que teria o raio como seu parâmetro, e o braço, que teria os parâmetros o raio interno, raio externo e ângulo de posicionamento. Por outro lado, uma transformação é uma modificação automatizada e semanticamente correta de um programa. As regras de reescrita ou regras de transformação constituem a categoria mais conhecida de transformação.

Um processo para a construção de geradores de aplicação é proposto por Cleaveland (1998), e compreende sete passos:

1) *Reconhecendo domínios:* Consiste em reconhecer quando um domínio é propício para a construção de geradores de aplicação. A regra básica é

reconhecer padrões tanto em nível de código (rotinas similares, código repetitivo, etc.) quanto em alto nível (programas similares, arquiteturas, projetos, etc.).

2) *Definindo os limites do domínio:* Este passo define a cobertura do domínio do gerador. Aqui devem-se decidir quais aspectos do domínio serão incluídos no gerador. Uma cobertura grande torna o gerador mais abrangente, mas provavelmente menos eficiente ou mais difícil de usar. Uma cobertura pequena traz eficiência, porém para uma limitada gama de problemas.

3) *Definindo um modelo básico:* Consiste em prover um modelo matemático que irá tornar o gerador mais apto a ser compreensível, consistente e completo. Esse modelo torna-se a base para definir a semântica do domínio, e servir como referência para as características, a informação de especificação e o projeto do produto. Exemplos podem ser: conjuntos, grafos dirigidos, sistemas de lógica formal e modelos computacionais como as máquinas de estado finito.

4) *Definindo as partes variantes e invariantes do domínio:* Decidir quais aspectos da geração poderão ou não ser parametrizadas pelo usuário. A parte variante é a que pode ser modificada pelo desenvolvedor; a parte invariante ou fixa é a que não pode ser configurada.

5) *Definindo a especificação de entrada:* Consiste em definir o modo, dentre os já mencionados (textual, gráfico, interativo, etc.) pelo qual o usuário irá especificar cada instância de programa a ser gerado.

6) *Definindo produtos:* Consiste em definir quais produtos o gerador irá produzir (exemplos: programas, documentação, dados de teste, etc.), qual abordagem de geração de código será usada (exemplos: baseadas em código, baseadas em tabelas), linguagem-alvo (no caso de geração de código) e se haverá algum recurso de otimização para reduzir requisitos de tempo e espaço.

7) *Implementando o gerador:* Consiste em desenvolver um programa que traduz a especificação de entrada no(s) produto(s) desejado(s).

2.4.2.4. Arquitetura Básica de um Gerador de Aplicação

Para projetar um gerador, precisa-se conhecer a arquitetura genérica dos mesmos. Geradores de aplicações têm grande semelhança com compiladores (AHO, RAVI, ULLMAN, 1995) que, descritos de forma simples, são programas

que lêem um programa escrito numa linguagem (a linguagem-fonte) e traduzem-no para um programa equivalente numa outra linguagem (a linguagem alvo), relatando também ao usuário a presença de possíveis erros no programa fonte. No caso do gerador de aplicações a linguagem-fonte é a Linguagem Específica de Domínio (LED).

Desta forma, a arquitetura básica de um gerador é similar a de um compilador (Figura 2.4) e é estruturada em três elementos principais:

- Módulo primário (*front-end*): responsável principalmente pela análise léxica, fazendo um mapeamento da forma de entrada original para uma representação interna intermediária, possível de automatização, como grafos de fluxo e árvores sintáticas;
- Mecanismo de tradução: parte principal do gerador que implementa transformações na representação intermediária (RI), produzindo um programa, mas ainda representado como grafos de fluxo ou árvores sintáticas;
- Módulo secundário (*back-end*): faz o mapeamento da RI do programa gerado para o texto do programa na linguagem-alvo.

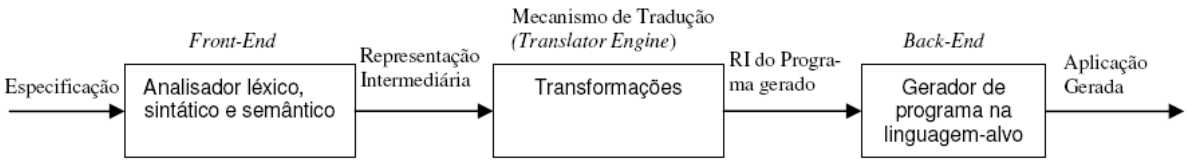


Figura 2.4: Arquitetura básica de um gerador.
(Adaptação de AHO, RAVI, ULLMAN, 1995).

O que difere um gerador de aplicação de um compilador é que suas especificações de entrada são de mais alto nível, normalmente são abstrações de um domínio de aplicação limitado; e sua expansão de código (razão do tamanho na entrada em relação ao tamanho da saída) é bem maior que no caso dos compiladores (KRUEGER, 1992). Outra diferença notável é que os compiladores produzem diretamente instruções executáveis, enquanto os geradores, na maioria dos casos, produzem somente uma transformação em formato diferente (não diretamente executável).

2.4.2.4. Formas de Geração de Código

Os geradores de código dividem-se em duas categorias: geradores ativos e passivos (HERRINGTON, 2003). Os geradores passivos geram código que fica disponível para ser editado ou alterado pelos desenvolvedores, não mantendo nenhuma responsabilidade pelo código gerado, nem a curto e longo prazo. Os geradores passivos são usados apenas uma vez para criar algum artefato. Concluída a geração, o desenvolvedor assume o item gerado, melhorando-o quando necessário. Os *wizards* existentes em Ambientes Integrados de Desenvolvimento são exemplos de geradores passivos.

Os geradores ativos mantêm responsabilidade pelo código gerado, permitindo diversas gerações da saída a partir de mudanças na sua entrada durante a evolução do projeto, tornando-se, portanto, parte do processo de desenvolvimento. A partir de um modelo como entrada, o gerador transforma-o num artefato diferente. Se o modelo for modificado, o processo de geração deve ser repetido. Normalmente, não é permitido ao desenvolvedor alterar o código gerado, ou é utilizado algum padrão eficiente para distinguir o código gerado do código escrito "à mão". A seguir serão apresentadas formas de geração ativa de código, de acordo com Herrington (2003).

1) "Code Munging"

Uma das formas mais comuns é a técnica conhecida como "*code munging*", que consiste em "moldar" a forma de entrada para obter a saída desejada. O termo "*munging*" nessa nomenclatura é uma gíria para torcer e moldar. O processo dessa geração (Figura 2.5) é bem simples e consiste em tomar um código fonte como entrada, geralmente usando expressões regulares ou *parsing* de código, e construir os arquivos de saída, usando *templates* internos ou externos. Esse tipo de geração pode ser usado para criar documentação ou ler constantes ou protótipos de funções a partir de um arquivo. Um exemplo é o gerador de documentação de programas Java (JavaDoc), que lê os comentários nos códigos fonte Java e gera uma documentação HTML a partir dos comentários, usando alguns *templates*; outro exemplo é o XDoclet, que gera classes EJB e interfaces necessárias para apoiar o *bean* que contém os comentários.

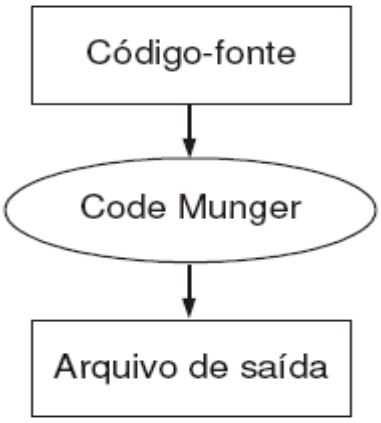


Figura 2.5: Processo de um gerador tipo *"code munging".*
(Adaptação de HERRINGTON, 2003).

2) "Inline code expander"

O "*Inline Code Expander"* ou expansor de código em linha tem como entrada um código fonte contendo marcações especiais que o *expansor* troca pelo código de produção para criar o código de saída, como mostrado na Figura 2.6. São geralmente usados para embutir comandos SQL em um código fonte, com marcas especiais no código SQL que serão reconhecidas pelo *expansor* que, ao ler o código, insere o código fonte que implementa a consulta ou comando SQL no ponto das marcações, tais como, Pro*C e SQLJ.

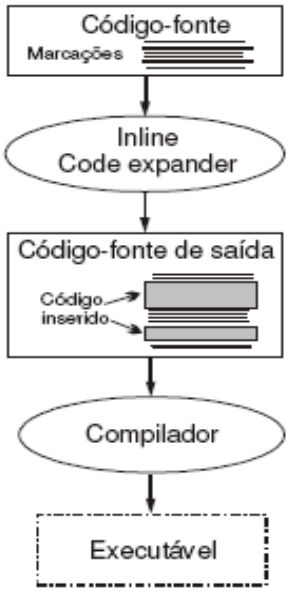


Figura 2.6: Processo de um gerador tipo *"inline code expander".*
(Adaptação de HERRINGTON, 2003).

Outros usos desse tipo de gerador incluem a inserção de seções de montagem de performance crítica e inserção de equações matemáticas que são implementadas pelo gerador. A principal vantagem é ser uma solução fácil para a simplificação do código-fonte e a desvantagem é a dificuldade de depurar o código de saída, que será bem maior que o código original.

3) "mixed-code generator"

Um gerador de código misto ("*mixed-code generator*") lê um arquivo de código fonte de entrada, procurando por comentários específicos, preenchendo esses pontos com código, substituindo o arquivo original. A diferença do expansor de código em linha é que o gerador de código misto insere o código diretamente no código de entrada (Figura 2.7).

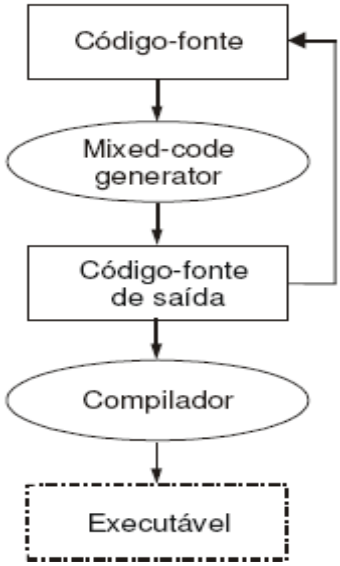


Figura 2.7: Processo de um gerador tipo *"mixed-code"*
*(*Adaptação de HERRINGTON, 2003).

Pode ser usado para implementar o mapeamento de controles da interface do usuário para variáveis de estrutura de dados, criar casos de teste a partir de dados de teste inseridos nos comentários e também para inserção de código de acesso a dados.

4) "Partial-class generation"

Um gerador de classes parcial, ilustrado na Figura 2.8, lê um arquivo com uma definição abstrata e, baseado em *templates*, constrói um conjunto de bibliotecas

de classes-base. Essas classes então são compiladas juntamente com as classes derivadas produzidas pelos engenheiros para completar o conjunto de produção de classes.

É um dos primeiros geradores a criar código a partir de uma especificação abstrata. Os modelos anteriores usavam código fonte em alguma linguagem (como C, Java, C++ ou SQL) como entrada. Em vez de inserir ou trocar fragmentos por meio da filtragem do código de entrada, este modelo recebe uma descrição de código e cria um conjunto completo de código implementado. Esse tipo de geração é um bom ponto de partida para construir um gerador que cria uma camada inteira de código.

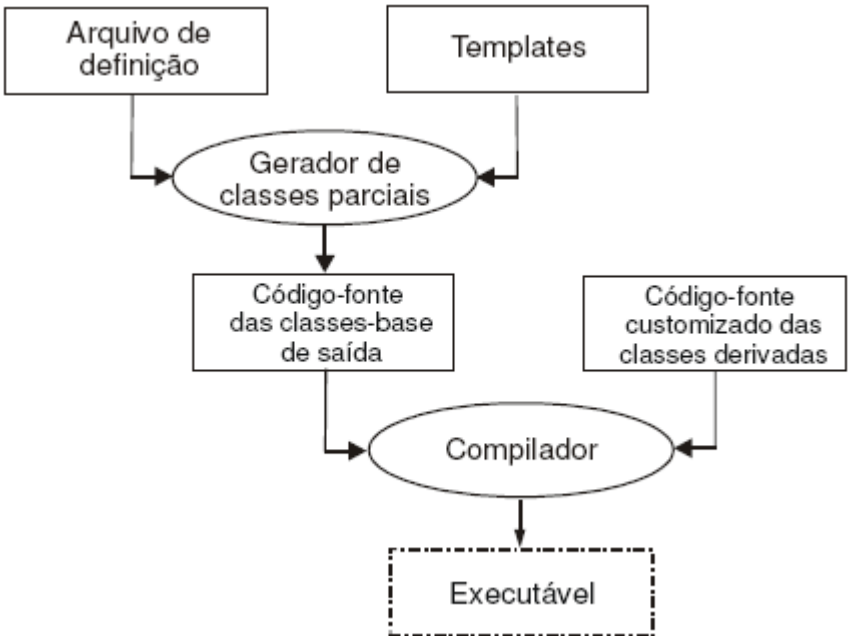


Figura 2.8: Processo de um gerador de classes parcial.
(Adaptação de HERRINGTON, 2003).

5) Geração de camada

Na geração em camadas ("*tier generation*"), o gerador tem a função de gerar uma camada completa de uma aplicação multicamada. O processo é o mesmo existente na geração de classes parciais: o gerador lê um arquivo de especificação e usa templates para gerar classes de saída que implementam o que está definido no arquivo de entrada, conforme Figura 2.9. A diferença é que neste caso o gerador constrói todo o código de uma camada, enquanto na

geração parcial as classes-base precisam ser complementadas com classes derivadas para completar o código da camada.

Um exemplo desse tipo de geração é a geração dirigida a modelo (*modeldriven generation*), na qual uma aplicação definida em UML e uma especificação de entrada em XML são usadas pelo gerador para construir uma ou mais camadas de um sistema. Outros exemplos de uso incluem a geração de camadas de *stored procedures* para acesso a banco de dados e a geração de camadas de importação, exportação ou conversão.

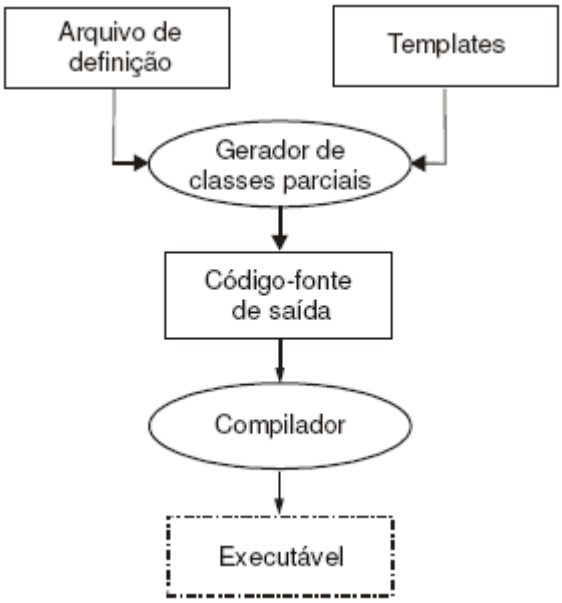


Figura 2.9: Processo de um gerador de camada.
(Adaptação de HERRINGTON, 2003).

2.5. Ferramentas

Vários trabalhos (BARRÉRE, 1999; PRADO, LUCRÉCIO, 2001; PAIS et al., 2001; PAZIN, 2004; GREVE et al, 2004; ROCHA et al, 2004; ANDROMDA, 2006; SHIMABUKURO, 2006) foram desenvolvidos a fim de gerar código de forma automática e eficiente, contribuindo assim com um ganho de produtividade por parte do engenheiro de software. Na Tabela 2.1 é ilustrado um quadro resumo comparando as principais características de algumas ferramentas encontradas na literatura.

Tabela 2.1: Exemplos de ferramentas que geram código.

Nome	Características	Origem	SW Livre?	Tecnologias utilizadas	Código gerado	Bibliografia
Cordel	Ferramenta para geração de sistemas *Web*	UFBA	Sim	Arquitetura baseada em MDA, utilizando Struts, J2EE, UML e XML	J2EE	(GREVE et al, 2004)
Xspeed	Ferramenta para geração de aplicações distribuídas	UFCE	Sim	UML, XMI, XML	Java	(ROCHA et al, 2004)
Code Charge	Gerador de código para *WebApps*	*Yes Software Corporation*	Não	-	ASP e JSP	(SANTOS, 2002)
GawCRe	Gerador de aplicações baseados na *Web* para sistemas de gestão em clínicas de reabilitação	UFSCar	Sim	LMA, XML, *framework GEEN, Java*	Classes Java (Beans) e interfaces em JSP	(PAZIN, 2004)
AndroMDA	Gerador de código a partir de especificações em UML	Andromda.org	Sim	Especificaçõe s em UML e ferramentas Hibernate, Springs e SOAP	Java	(ANDROMDA, 2006)
MVCASE	Ferramenta CASE que suporta a especificação, análise e implementação de requisitos de sistemas de software orientados a objetos	UFSCar	Sim	JavaRC, Fusion e UML	Java/ EJB e XML	(BARRÉRE, 1999; PRADO, LUCRÉDIO, 2001; LUCRÉDIO et al, 2003)
Hercules	Um arcabouço para desenvolvimento de sistemas de informação, visando possibilitar a geração automática de código a partir de diagramas UML	UFRJ	Sim	UML e Java	Java	(PAIS, OLIVEIRA, LEITE, 2001)
Captor	Gerador de aplicações configurável para fornecer apoio em domínios específicos	ICMC-USP São Carlos	Sim	LMA, XML, *Java*	Classes Java (Beans) e interfaces em JSP	(SHIMABU-KURO, 2006)

A ferramenta *Cordel* (GREVE et al, 2004) promove de forma flexível a geração, compilação e implantação automática de sistemas *Web* a partir de modelos UML. O Cordel utiliza o conceito de *flavour*, que são conjuntos de tecnologias que definem a arquitetura a ser implementada. Inicialmente, a

ferramenta possui um *flavour* padrão que implementa aplicações J2EE com as seguintes características: *framework* Struts e páginas JSP para camada de apresentação, servidor EJB JBOSS para camada de aplicação, e bancos de dados PostgreSQL ou HyperSonic. Além da arquitetura padrão o usuário pode anexar à ferramenta outros *flavours*, de modo a personalizar a geração de código para outras arquiteturas.

A ferramenta *AndroMDA* (ANDROMDA, 2006) permite gerar código fonte para a plataforma Java a partir da especificação UML. A ferramenta sugere o uso de diversas tecnologias para persistência dos dados como *Hibernate, Spring* e *SOAP*, ao mesmo tempo em que se propõe a diminuir o tempo de desenvolvimento de programas e promover o uso intensivo de padrões.

A *Xspeed* (ROCHA et al, 2004) é uma ferramenta que objetiva receber como entrada um arquivo no formato XMI (*XML Metadata Interchange*) contendo o modelo UML da aplicação a ser gerada. Em seguida, mapeia os padrões que deverão ser aplicados como estratégia para resolução de problemas e finalmente faz a geração automática do código interligando as tecnologias específicas para a resolução de cada um dos problemas inerentes ao domínio.

O *GawCRe* (PAZIN, 2004) é um gerador de aplicação baseado na *Web* para sistemas de gestão em clinicas de reabilitação. Foi implementado utilizando os conceitos de linha de produtos de software e uma arquitetura representada por uma linguagem de modelagem de aplicações (LMA), definida com base na linguagem de padrões denominada GRN (Gestão de Recursos de Negócios), desenvolvido utilizando um meta-modelo e a linguagem XML.

O CodeCharge (SANTOS, 2002) é uma ferramenta comercial que integra a função de geração de código para aplicativos de acesso a banco de dados via Web, sendo possível criar aplicações Web nas linguagens *Active Server Pages* (ASP) e *Java Server Pages* (JSP).

A MVCASE (BARRÉRE, 1999; PRADO, LUCRÉDIO, 2001, LUCRÉDIO et al, 2003) é uma ferramenta CASE orientada a objetos que suporta a especificação de requisitos do sistema usando técnicas UML. Permite a construção de modelos gráficos e textuais que representam o sistema em diferentes níveis de abstração. Os modelos gráficos de uma especificação na MVCASE facilitam a comunicação

entre os desenvolvedores do sistema e seus usuários. A MVCASE pode gerar o código Java/EJB dos componentes. Além do código Java/EJB, a ferramenta gera as descrições em XML em um servidor EJB para serem reutilizadas pelas diferentes aplicações.

Uma outra implementação é realizada pelo projeto Hércules (PAIS et al., 2001), que consiste em um arcabouço para desenvolvimento de sistemas de informação, visando possibilitar a geração automática de código a partir de diagramas UML, e estabelece um conjunto de descrições de alto nível de abstração, que os analistas ou programadores utilizam para especificar a apresentação, o desenvolvimento operacional e a persistência do sistema a ser construído. Este modelo é dividido em três camadas: domínio, controle e apresentação. Cada uma destas camadas possui diagramas UML associados.

O Captor, (SHIMABUKURO, 2006), é um gerador de aplicação configurável desenvolvido na linguagem Java, que pode ser configurado para diversos domínios e fornecer apoio no desenvolvimento de diversas aplicações dentro de um mesmo domínio. Para utilizar o Captor, o Engenheiro de Domínio deve configurá-lo e o Engenheiro de Aplicação deve utilizá-lo para a construção de aplicações em um domínio particular. A configuração é realizada por meio da modularização da especificação em formulários ou, mais especificamente, criação de linguagens de modelagem de aplicações baseadas em formulários. Cada formulário apresenta um conjunto de campos, representados por elementos gráficos, tais como: caixas de texto, caixas de seleção, tabelas, entre outros. A interface gráfica com o usuário (GUI) do Captor é configurada para apresentar comportamento variável, ou seja, ela é configurada para modificar a sua aparência para receber especificações baseadas em formulários de diferentes domínios.

Apesar da existência de várias ferramentas disponíveis para geração de aplicações, das quais algumas como o Captor poderiam ser utilizadas neste projeto mesmo que de forma experimental, enfatizando ainda mais a técnica de reuso, optou-se pelo desenvolvimento do próprio gerador de aplicações da ferramenta Fênix, integrado ao *framework* Titan. Tal decisão ocorreu motivada em dar continuidade do trabalho já em andamento, buscando domínio e consolidação

da tecnologia neste domínio no LEDES e, pela constatação da necessidade de alteração no projeto para adaptação e uso de algum gerador já existente, o que, invariavelmente, levaria a desgastes extras.

2.6. Considerações Finais

Neste capítulo foi apresentada uma visão sobre os principais conceitos e técnicas relacionadas ao reuso de software, como o uso de componentes, *frameworks* e LPS. O uso de geradores de aplicação foi enfatizado, pois é a principal motivação para a realização deste trabalho. No próximo capítulo será apresentada a ferramenta Fênix e o *framework* Titan, que são infra-estruturas básicas para implementação do gerador de aplicação proposto.

Capítulo 3
Ferramenta Fênix

3.1. Considerações Iniciais

A concepção da ferramenta Fênix (CARROMEU, TURINE, 2005, 2006a, 2006b; CARROMEU, 2007) foi objeto de outro trabalho de mestrado em Engenharia de Software no DCT-UFMS. Tem como objetivo instanciar e gerar *WebApps* no domínio de sistemas e-Gov a fim de auxiliar a gestão (submissão, avaliação, monitoramento e finalização) de propostas eletrônicas de fomento de projetos a serem avaliadas por agências de fomento no Brasil. A ferramenta automatiza um processo de desenvolvimento definido por uma LPS orientada a família de produtos no domínio de SAGF.

A partir da ferramenta Fênix objetiva-se desenvolver aplicações de forma rápida e eficaz que possam prover a transparência das informações e a desburocratização de ações no gerenciamento de projetos submetidos a agências de fomento, automatizando o processo de gerência de projetos desde a fase de submissão de propostas eletrônicas a serem avaliadas por agências governamentais, acompanhamento técnico e administrativo.

Neste capítulo serão apresentadas as principais funcionalidades e a arquitetura da ferramenta Fênix, destacando o subsistema gerador de aplicação desenvolvido neste trabalho. Na Seção 3.2 são apresentadas a arquitetura da ferramenta e as tecnologias de software livre utilizadas, tais como, Java (JSP, Struts e Hibernate), XML e PostgreSQL; na Seção 3.3 é apresentado o Titan, um *framework* que permite a instanciação de WebApps no domínio de sistemas gerenciadores de conteúdo e foi utilizado no processo de geração de código e, por fim, na Seção 3.3 as considerações finais do capítulo.

3.2. Arquitetura da Ferramente Fênix

O processo de desenvolvimento subjacente à ferramenta Fênix é baseado, principalmente, na abordagem orientada à família de produtos FAST (*Family-Oriented Abstraction, Specification, and Translation*) (LAI, WEISS, 1999), seguindo três passos, conforme ilustrado formalmente[2] na Figura 3.1: (1) Qualificação da Família Fomento, (2) Engenharia do Domínio da Família Fomento e (3) Engenharia da Aplicação da Família Fomento, sendo que nos dois últimos passos o processo é iterativo e incremental. O passo 1 compreende a determinação dos objetivos e dos requisitos de negócio do domínio. A Engenharia do Domínio (passo 2) compreende a especificação dos diferentes artefatos de reutilização que definem a arquitetura da LPS. A partir destes artefatos é criado o Ambiente de Engenharia da Aplicação, composto pelo *framework* Titan, os modelos de análise do domínio da família (modelos de *features*, de casos de uso e de classes) e a ferramenta Fênix que automatiza o processo de engenharia da aplicação. Na Engenharia da Aplicação (passo 3), a partir de um processo iterativo, o engenheiro refina os artefatos do Ambiente da Engenharia da Aplicação e modela/instancia diferentes aplicações no domínio SAGF.

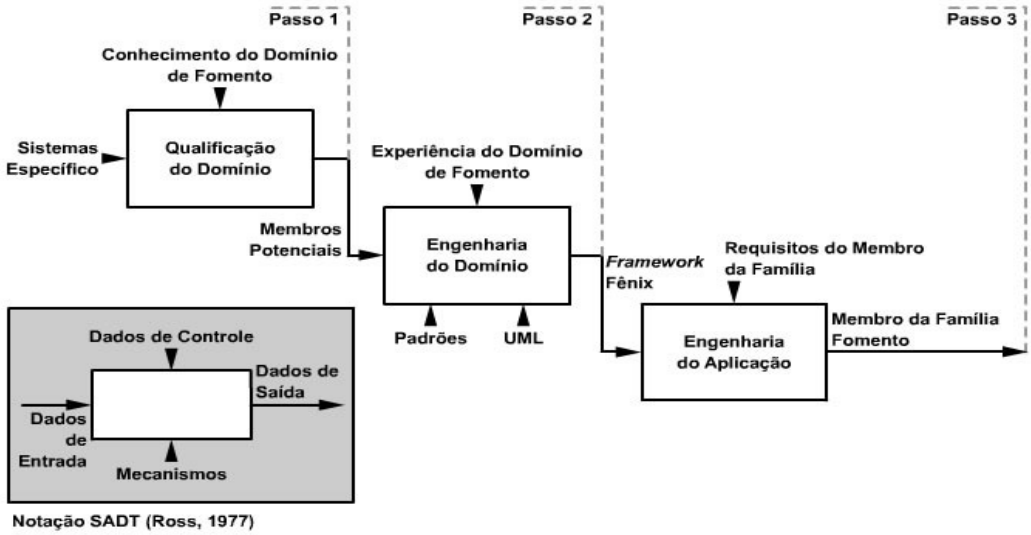


Figura 3.1: Processo formal da LPS de SAGF.(CARROMEU, 2006).

[2] Para definir formalmente o processo de desenvolvimento da LPS em que se incluem as fases e os artefatos utiliza-se a notação de Roos (ROOS, 1977).

A partir da abordagem de desenvolvimento de frameworks baseada em exemplos (JOHNSON, 1992) e as aplicações Fundect OnLine (http://www.fundect.ms.gov.br) e SIEX-IPES (http://siex.ledes.net) e JEMS (https://submissoes.sbc.org.br), foram abstraídos os padrões do domínio. Em seguida, cada padrão foi mapeado na forma de *features* (COHEN et. al, 1990). Assim, foram especificados quatro padrões básicos de gestão na ferramenta:

- Formulários: são responsáveis pela inserção de dados pelos usuários da aplicação instanciada;
- Atributos: são elementos independentes que podem ser utilizados nos formulários da aplicação instanciada;
- Grupos de Usuários: são os grupos passíveis de interação com a aplicação instanciada; e
- Módulos: são os *workflows* de submissão e gerência de propostas eletrônicas. Mais especificamente, os módulos são responsáveis por integrar de forma interativa Grupos de Usuários e Formulários em diferentes momentos da aplicação instanciada.

A arquitetura da LPS, subjacente à ferramenta Fênix, é baseada em dois sistemas autônomos: um sistema gerador e um sistema gerado (CARROMEU; TURINE, 2005, 2006a, 2006b; CARROMEU, 2007). O sistema gerador é, portanto, a ferramenta Fênix em si, enquanto o sistema gerado será a instância do *framework* Titan, ou seja, o sistema *Web* de apoio a fomento que o Fênix se propõe a instanciar.

A arquitetura da ferramenta Fênix é ilustrada na Figura 3.2., sendo subdividida em três subsistemas: (1) Sistema de Segurança, (2) Sistema de Gestão e (3) Gerador de Aplicação. O Sistema de Gestão compõe-se de quatro gestores, por meio dos quais o usuário poderá definir e configurar todo o sistema gerado: Gestor de Grupos de Usuários, Gestor de Módulos, Gestor de Formulários e Gestor de Atributos. Estes gestores podem ser considerados como ferramentas de configuração dos *features* do *framework*. O Sistema de Segurança responde pela camada de autenticação do sistema gerador e o Gerador de Aplicação é responsável por gerar, de modo interativo, o sistema gerado. Cada

um destes sistemas e gestores será abordado mais profundamente nas próximas seções.

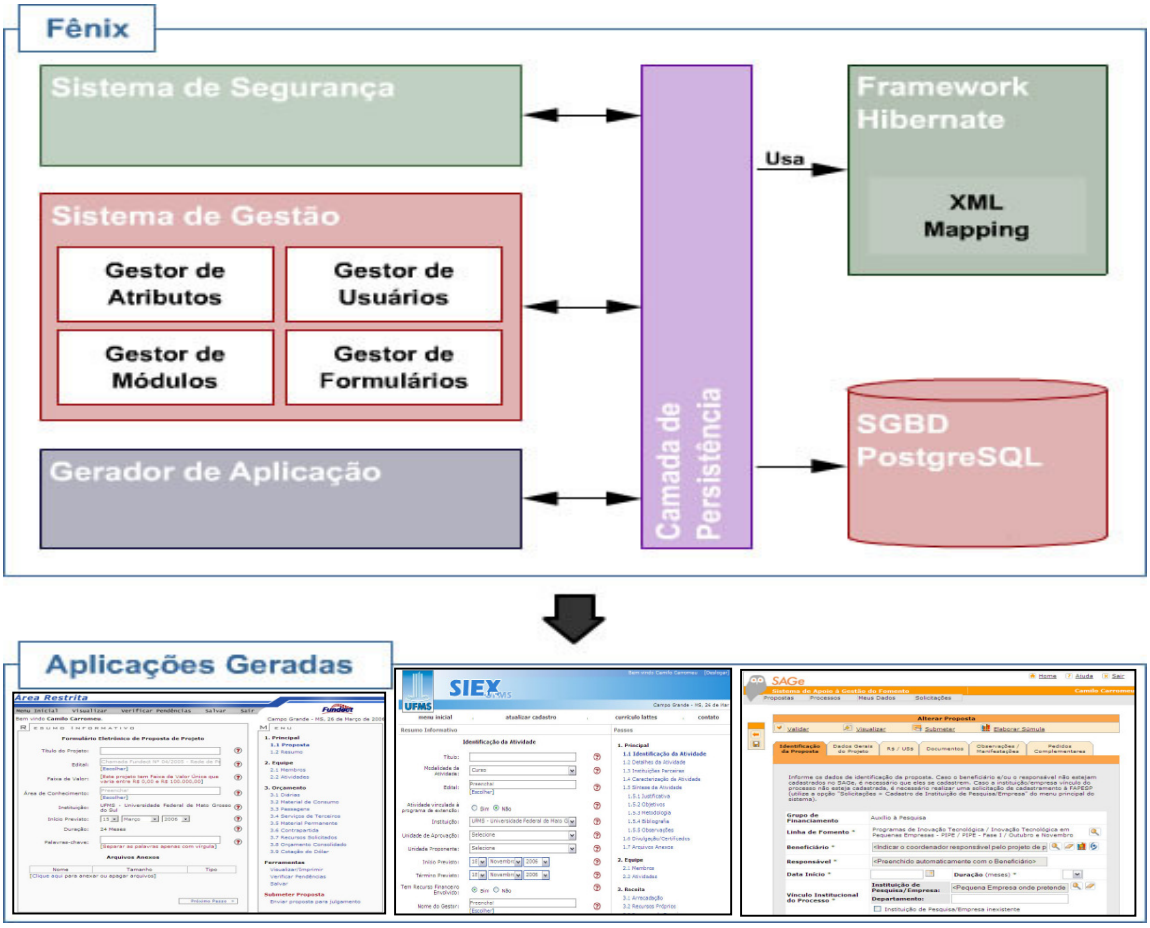


Figura 3.2: Arquitetura da ferramenta Fênix. (CARROMEU, 2007. p. 69)

Em relação às tecnologias utilizadas, a arquitetura do Fênix é dividida em três camadas, seguindo os conceitos definidos no padrão MVC (*Model-View-Controller*) (STEARNS et all, 2002). A camada de apresentação é responsável pela interface gráfica com o usuário e foi implementada usando JSP. Essa camada gerencia a entrada e a saída dos dados, comunicando-se com a camada de negócio para obter os dados a serem apresentados na interface com o usuário e para recuperar informações e realizar solicitações que serão processadas pela camada de negócio. A camada de negócio é formada por *beans* e classes de ação que manipulam a criação de objetos. As classes de ação foram implementadas a partir do *framework* Struts (HUSTED et all, 2003). Essa camada comunica-se com a camada de persistência quando há necessidade armazenar

um objeto permanentemente, listar ou alterar objetos persistentes. A camada de persistência utiliza o *framework* Hibernate (BAUER, KING, 2005) para realizar toda a interação com a base de dados. Utiliza-se o sistema de banco de dados PostgreSQL (STINSON, 2001), entretanto uma das características do Hibernate é tornar a aplicação portável para diversos SGBDs.

3.2.1. Sistema de Segurança

O Sistema de Segurança constitui um requisito não funcional e é responsável pela segurança das informações e funcionalidades da ferramenta Fênix. O mecanismo de segurança é realizado por um subsistema de controle de acesso que gerencia as permissões de grupos de usuários sobre as funcionalidades e informações da ferramenta; e por um subsistema de autenticação, que é responsável por autenticar os usuários do sistema.

O cadastro de novos usuários no sistema é feito por usuários que pertençam a grupos que tenham tal permissão. Existe um grupo de usuários inicial e permanente chamado Administrador. Usuários pertencentes ao grupo Administrador possuem todas as permissões possíveis do Fênix. A autenticação do usuário será feita por meio de um identificador (login) e uma senha, ambos definidos durante o cadastro do usuário. A senha é criptografada, padrão SHA-1 (*Secure Hash Standard*) (W3C, 1998), e armazenada no banco de dados. Assim, durante o processo de autenticação, a senha fornecida pelo usuário é criptografada antes de ser comparada com a informação armazenada no banco de dados.

Além do cadastro de usuários, o Sistema de Segurança do Fênix também gerencia grupos de usuários. Somente a grupos podem ser atribuídas permissões primárias do sistema, sendo que os usuários herdam tais permissões dos grupos a que pertencem. Estas permissões estão diretamente relacionadas a funcionalidades específicas do sistema e quando são atribuídas a um grupo pode-se ou não indicar a qual aplicação exatamente a permissão será concedida. Cada usuário cadastrado deverá estar vinculado a um ou mais grupos para herdar as respectivas permissões. A lista de permissões possíveis varia de acordo com as funcionalidades como, por exemplo, criar aplicações, cadastrar usuários e criar atributos.

3.2.2. Sistema de Gestão

O Sistema de Gestão da ferramenta Fênix permite a configuração dos padrões do *framework* Titan. O primeiro passo no processo de instanciação do Fênix é a criação de uma nova aplicação, que será uma instância do *framework.* O segundo passo é a configuração da aplicação, que consiste em definir os módulos da aplicação, seus formulários (interfaces gráficas com o usuário), conjunto de atributos necessários a cada interface e os grupos de usuários da aplicação. Na Figura 3.3 é apresentada a interface do Fênix, que contém um Menu que direciona o usuário a realizar todas essas configurações, ou seja, criar, editar e apagar novos Módulos, Grupos de Usuários, Atributos e Formulários para uma aplicação a ser instanciada.

Dessa forma, o sistema de Gestão da ferramenta Fênix gerencia o funcionamento de quatro outros subsistemas gestores: Gestor de Grupos de Usuários, Gestor de Atributos, Gestor de Formulários e Gestor de Módulos. Cada um destes gestores possibilita a configuração dos quatro padrões básicos do *framework*.

Figura 3.3: Menu de seleção de gestores de configuração. (CARROMEU, 2007. p.72).

3.2.2.1. Gestor de Grupos de Usuários

O Gestor de Grupos de Usuários é responsável pela criação e configuração de todo o sistema de segurança, autenticação e controle de acesso às aplicações

instanciadas a partir do Fênix. Tem como alvo a aplicação instanciada, enquanto o Sistema de Segurança do Fênix, conforme descrito anteriomente, tem como alvo o próprio Fênix.

O Gestor de Grupos de Usuários funciona de forma similar ao Sistema de Segurança do Fênix, permitindo criar grupos de usuário aos quais são atribuídas permissões de acesso à aplicação instanciada. Uma lista pré-determinada de permissões de variados tipos pode ser combinada em cada grupo em função de módulos que esses grupos podem ter acesso e do nível de acesso. O nível de acesso em um módulo pode ser medido pelas permissões que um usuário possui sobre os formulários pertencentes a este módulo. A combinação de um formulário, grupo de usuários e um estado do módulo formam uma "visão", conforme definido pelo Gestor de Módulos.

3.2.2.2. Gestor de Módulos

Um módulo representa os diferentes tipos de projetos que deverão ser gerenciados pela aplicação gerada, por exemplo, propostas de projetos de pesquisa submetidos à avaliação de uma agência de fomento. Todo o trâmite, desde a submissão da proposta, sua avaliação, seu acompanhamento técnico e administrativo serão gerenciados por meio de uma série de passos, ou seja, os estados desta proposta.

A arquitetura genérica de um módulo envolve três abstrações: estado (Ei), transição de estados e visão do sistema. Cada módulo possui um conjunto finito de estados, (E0, E1, ..., Ef-1, Ef onde E0 representa o estado inicial e Ef o estado final) que representa o fluxo de execução das tarefas de um módulo (*workflow* linear da vida do módulo). A transição de um estado Ei para Ei+1 é dada pela ocorrência de um evento (gatilho), no qual alguns requisitos devem ser atendidos. Uma transição de estados é representada na Figura 3.4 por uma seta partindo do estado anterior e apontando para o estado seguinte. Um gatilho pode ser uma data específica, um botão em um formulário ou simplesmente um conjunto de requisitos atendidos. Em todos os casos, o sistema verifica se requisitos pré-determinados foram atendidos (exceto quando o cumprimento de tais requisitos define o próprio gatilho). No caso de uma submissão de proposta de projeto, a

transição de estados determina todo o ciclo de vida da proposta, desde o momento da submissão até o completo encerramento do projeto (ou não-aprovação do mesmo).

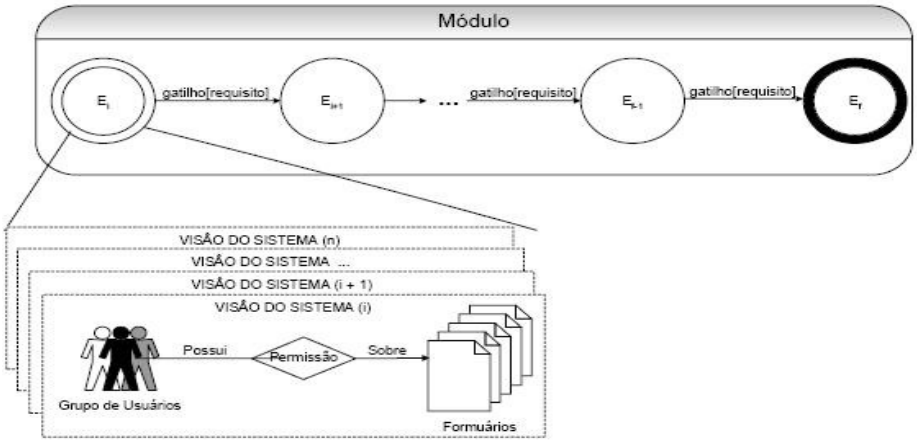


Figura 3.4: Arquitetura de um Módulo no Fênix.(CARROMEU, 2007. p. 74).

Cada estado do *workflow* de um Módulo agrupa um conjunto de "Visões" do sistema, sendo que cada visão constitui um relacionamento entre um Grupo de Usuários e um Formulário, ou seja, representa as permissões que cada Grupo de Usuários detêm sobre um determinado Formulário. Um mesmo grupo de usuários pode estar associado a diferentes formulários, formando diferentes visões no mesmo estado do *workflow*.

3.2.2.3. Gestor de Atributos

Um atributo é uma entidade que pode ser utilizada em diferentes momentos da aplicação e pode ser de vários tipos. Os tipos primários de atributos na ferramenta são: Integer, Date, String, Float e Boolean. Existem também tipos estendidos de atributos, que são variações dos tipos primários com funções mais específicas, tais como, CPF, CNPJ, Coin, Password e Text.

Os atributos são independentes das funcionalidades do sistema e isto torna possível a não restrição do seu uso. Atributos são utilizados na forma de campos de formulários e de variáveis de controle. Qualquer campo de formulário deve estar associado a um atributo (que terá seu valor modificado ou visualizado). Na forma de variáveis de controle, os atributos podem ser utilizados para verificação

de requisitos de mudança de estados em Módulos e na validação de campos de formulários através de equações lógicas. Assim, um atributo CPF de pesquisador, por exemplo, pode tanto ser utilizado no formulário de cadastro de pesquisador quanto no formulário de autenticação. Apesar de serem dois campos diferentes de dois formulários diferentes o atributo será o mesmo, ou seja, o mesmo valor cadastrado será utilizado para fazer autenticação no sistema.

3.2.2.4. Gestor de Formulários

Todos os dados necessários para a avaliação de uma proposta durante o seu ciclo de vida são inseridos por meio de formulários. Desde a submissão por meio de pesquisadores, as alterações e observações de funcionários da instituição de fomento, o julgamento de consultores, as revisões da equipe financeira e quaisquer outras informações relevantes que devem ser inseridas, modificadas ou visualizadas utilizam formulários. Formulários também são responsáveis pela inserção de usuários no sistema gerado (cadastros públicos, privados e protegidos).

O Gestor de Formulários possibilita a criação e edição de formulários para a aplicação. Na criação de um formulário devem, primeiramente, ser definidos os dados básicos do formulário (nome e descrição). Em seguida devem ser definidos três elementos que compõem a estrutura do formulário:

- Passos: cada formulário é composto por um ou mais passos. Os passos dividem o preenchimento do formulário em páginas, facilitando o preenchimento e aumentando a segurança (já que cada passo pode ser salvo independentemente);
- Grupos: a cada passo podem ser atribuídos um ou mais grupos. Cada grupo é, basicamente, um elemento organizador, sendo utilizado apenas para separar campos por contexto; e
- Campos: a unidade mais elementar de um formulário. Cada grupo, por sua vez, pode receber um ou mais campos. Um campo tem a função de instanciar um atributo e, desta forma, permitir sua visualização ou edição. Durante a criação do campo o usuário pode definir uma série de

características, tais como: nome, descrição, atributo que será instanciado e tipo do campo (visualização ou edição).

Um formulário poderá ser associado a uma visão de um determinado estado de algum módulo ou a um grupo de usuários, na forma de formulário de cadastro ou edição de dados cadastrais. Em uma visão, o formulário forma uma tríplice juntamente com um grupo de usuários e um estado do workflow de módulo.

3.2.3. Gerador de Aplicação

O Gerador de Aplicação é responsável por instanciar, de forma interativa, uma aplicação no domínio de SAGF. Basicamente, tem como objetivo recuperar toda a informação armazenada pelo Sistema de Gestão do Fênix e utilizá-la na criação do banco de dados, arquivos de configuração, metadados e código fonte do sistema gerado. No Fênix, todo o comportamento da aplicação a ser instanciada encontra-se devidamente armazenada, sendo da responsabilidade do Gerador de Aplicação recuperar tais informações e transformá-las em códigos da aplicação com toda a estrutura lógica de regras de negócios e física que o usuário projetou. A implementação deste gerador é objeto do presente trabalho e, portanto, será abordado de forma detalhada no próximo capítulo.

3.3. *Framework* Titan

O Titan (LEDES, 2006) é um *framework* de aplicações que foi especificado e implementado pelo Grupo de Pesquisa em Engenharia de Software do DCT-UFMS para auxiliar no desenvolvimento de WebApps no domínio de sistemas gerenciadores de conteúdo (CARROMEU, TURINE, 2006a). Por meio da estrutura de classes do Titan, é possível instanciar um gerenciador de conteúdo para páginas *Web*. Este gerenciamento permite criar, editar e apagar dados em SGBDs (Sistemas de Gerenciamento de Bancos de Dados) ou arquivos do sistema. Deve prover um sistema de segurança e autoria para as ações dos usuários sobre estes dados e um sistema de *log* com registro fiel destas ações.

A arquitetura do Titan segue o paradigma da Engenharia Baseada em Componentes (GIMENES, HUZITA; 2005) e é caracterizado como caixa-branca. Assim, se uma determinada funcionalidade não pode ser instanciada pelo *framework* é possível criar um novo componente no repositório. A linguagem genérica de entrada se encarregará de garantir que a configuração deste novo componente possa ser realizada como nos demais. O desenvolvimento do gerador de código do Fênix envolveu, portanto, a extensão do *framework* Titan a fim de englobar os padrões do domínio de gestão de fomento de projetos. Por ser um *framework* do tipo caixa-branca este processo torna-se possível e sustentável.

3.4. Considerações Finais

Neste capítulo foi apresentada uma visão geral da ferramenta Fênix, além do contexto da presente proposta de um gerador de aplicação na referida ferramenta. O *framework* Titan especifica uma arquitetura que recebe, como entrada, o conteúdo produzido pelo gerador de código do Fênix e o transforma, em tempo de execução, em uma *WebApp* no domínio especificado. No próximo capítulo será apresentada a arquitetura do gerador de aplicação desenvolvido neste trabalho.

Capítulo 4
Proposta: Gerador de Aplicação

4.1. Considerações Iniciais

Conforme apresentado anteriormente, geradores de aplicações são ferramentas capazes de automatizar parte do processo de desenvolvimento de software, convertendo especificações de alto nível em artefatos de software. Dessa forma, podem ser considerados como compiladores para uma linguagem de um domínio específico (SMARAGDAKIS; BATORY, 1998). Uma linguagem de domínio pode ser representada por meio de uma linguagem de padrões, uma Linguagem de Modelagem de Aplicações (LMA, originalmente *AML – Application Modeling Language*). As especificações escritas em qualquer um destes processos devem representar modelos, isto é, devem ser abstrações das aplicações que garantam certas propriedades importantes para o domínio (LAI, WEISS, 1999).

Para a construção de um gerador de aplicação deve-se obter a estrutura comum e variável do domínio alvo. O modelo de domínio pode auxiliar na fase de análise das similaridades desses sistemas, mas geralmente os estudos de suas variabilidades não podem ser feitos de forma sistemática. As variações podem ser entendidas por meio do conhecimento do Engenheiro de Domínio sobre o domínio desejado. Torna-se necessário entender as similaridades e as variabilidades para que se possa projetar uma ferramenta de geração robusta, que cubra o máximo possível do domínio. A necessidade de se modelar as similaridades e variabilidades do sistema surge naturalmente no decorrer da evolução do processo de desenvolvimento, e torna-se muito importante no reuso (JARZABECK, 1995).

Neste capítulo é apresentado um modelo de arquitetura para o gerador automático de WebApps baseado no *framework* Titan e integrado à ferramenta Fênix, que auxilia no processo de desenvolvimento de aplicações no domínio de

Sistemas Web de Apoio à Gestão de Fomento de Projetos. A Seção 4.2 apresenta as diretrizes para a especificação da arquitetura do gerador a partir da ferramenta Fênix e do processo de LPS implementada e usada na mesma; na Seção 4.3 é explicado o processo de definição, estruturação e implementação do gerador. As tecnologias adotadas são abordadas na Seção 4.4 e as considerações finais são feitas na Seção 4.5.

4.2. O Gerador no Contexto do Fênix e LPS

O Gerador de Aplicações da ferramenta Fênix é um módulo especial da ferramenta, que tem a responsabilidade de gerar uma nova aplicação instanciada por meio do Sistema de Gestão.

Para concepção e modelagem da ferramenta foi especificada uma arquitetura genérica, ilustrada na Figura 4.1, que atendesse a todos os requisitos dos sistemas a serem gerados. Dessa forma, quatro etapas distintas e sucessivas, agrupadas em três passos são definidos, iniciando-se o processo pelos Requisitos da Aplicação, que compreende a determinação dos objetivos e dos requisitos de negócio do domínio da aplicação a ser instanciada (gerada) e que o Engenheiro de Aplicação deve estabelecer previamente. A seguir, essas informações são passadas pelo Engenheiro de Aplicação, passo (1), para o Sistema Gestor do Fênix, por meio da interface gráfica (formulários) da ferramenta. O módulo Sistema Gestor do Fênix tem a responsabilidade de coletar, refinar e armazenar os artefatos idealizados pelo Engenheiro da Aplicação e modelar a mesma no domínio SAGF a ser criada. O módulo Gerador de Aplicações do Fênix é responsável por instanciar, de forma interativa, uma aplicação no domínio de SAGF previamente definida e armazenada na base de dados do Fênix. Basicamente, este sistema irá recuperar, toda a informação armazenada pelo Sistema de Gestão, passo (2), e utilizá-la para criação do banco de dados, arquivos de configuração, metadados e código fonte do sistema gerado. A quarta etapa é representada pelo módulo Aplicação Gerada, e constitui-se nos produtos finais desenvolvidos e recebidos, passo (3), do módulo Gerador de Aplicações. Esta última etapa (passo 3) é realizada de forma automática, ou seja, sem a interferência ou ações do Engenheiro de Aplicação.

Após a finalização do processo de geração, o Engenheiro de Aplicação tem a oportunidade de avaliar o produto gerado. Caso a aplicação instanciada não esteja em conformidade com projeto idealizado inicialmente, faz-se necessário retorno ao módulo inicial para revisão dos Requisitos da Aplicação e conseqüente execução de todos os demais passos subseqüentes, repetindo, dessa forma, o processo, para que a aplicação seja gerada novamente.

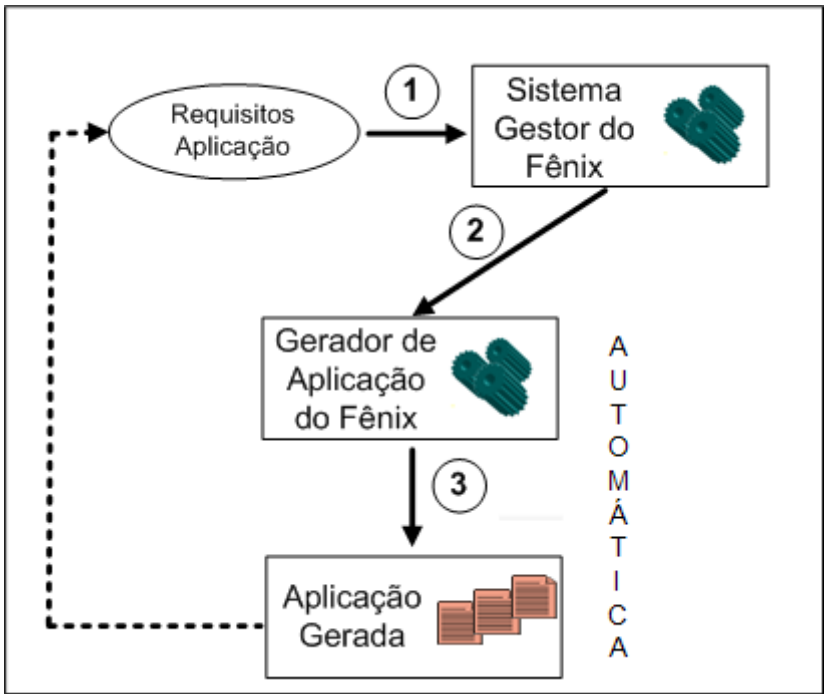


Figura 4.1: Visão geral da geração de aplicações no Fênix.

4.3. Arquitetura

O Gerador de Aplicação integra o Sistema de Gestão do Fênix e o *framework* Titan, e tem como objetivo principal automatizar o desenvolvimento de *WebApp* de apoio a gestão de fomentos, utilizando um SGBD relacional.

Na Figura 4.2 é ilustrada a arquitetura do gerador de aplicação da ferramenta Fênix integrado ao *framework* Titan. Resumidamente, o gerador de aplicação deverá fazer o mapeamento automático de uma especificação instanciada no Fênix, por meio do Sistema de Gestão, para a implementação de uma aplicação específica da família SAGF, validando a sua entrada no *framework* Titan e gerando um código fonte na linguagem-alvo.

Neste processo, tem-se a figura de dois atores: o Engenheiro de Aplicação, que interage diretamente com a ferramenta, utilizando-a para especificar e gerar aplicações por meio de uma interface gráfica com o usuário (GUI); e o Usuário Final, que utiliza e executa o código da aplicação gerada.

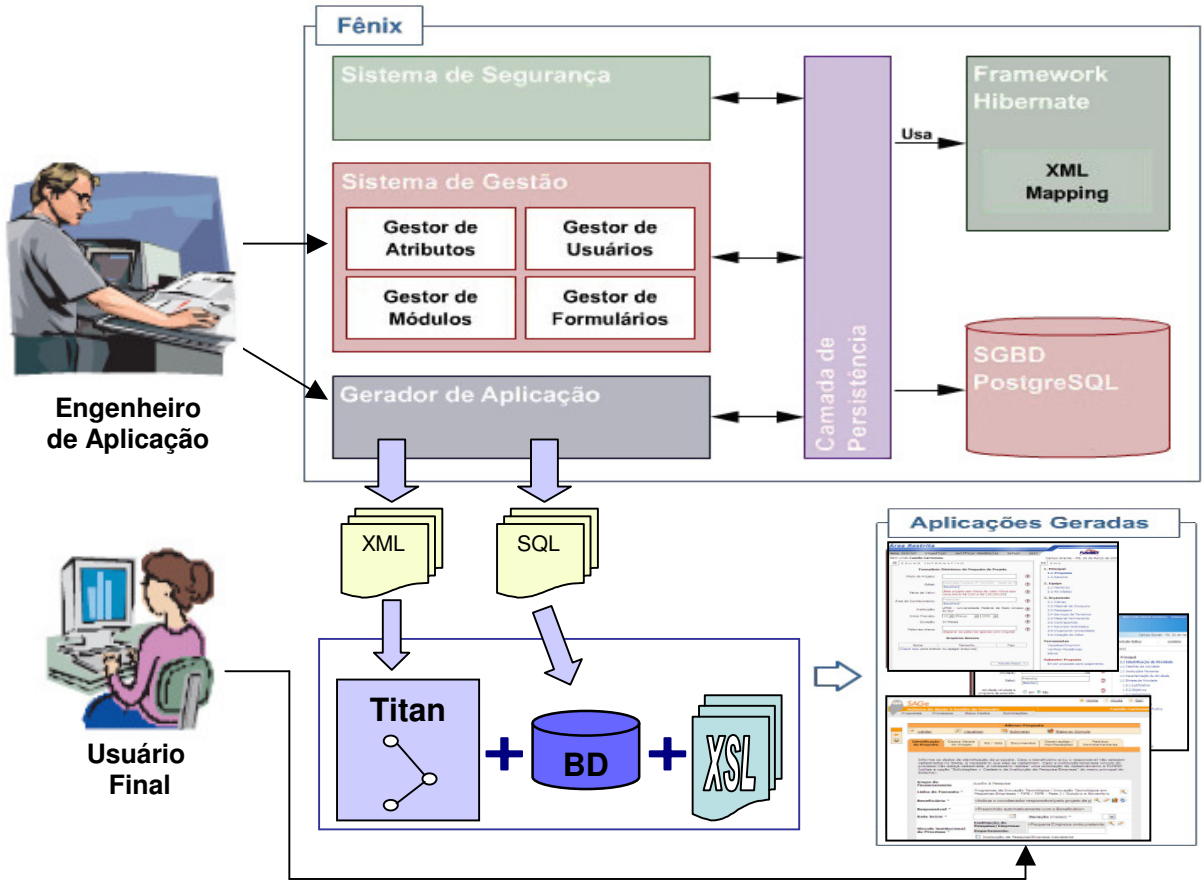


Figura 4.2: Arquitetura geral do gerador de aplicações.

Cada uma das duas etapas especificadas para o Engenheiro de Aplicação é realizada de forma independente, porém consecutivas, ou seja, é necessário que primeiro o Engenheiro execute e finalize as atividades de responsabilidade do módulo Sistema de Gestão, uma vez que os resultados desta etapa são utilizados como requisitos de entrada para o Gerador de Aplicação integrado com o *Framework* Titan, resultando dessa forma na aplicação instanciada e gerada, que fica disponível para ser utilizada pelo Usuário Final. Esse fluxo de execução está em consonância com o processo geral da LPS de SAGF e a arquitetura geral de passos para gerar aplicação no Fênix, visualizada na Figura 4.1 da seção anterior.

De acordo com a Figura 4.2, o Engenheiro de Aplicação deve utilizar inicialmente, o Sistema de Gestão do Fênix (detalhado no Capítulo 3), composto por quatro gestores: Gestor de Grupos de Usuários, Gestor de Módulos, Gestor de Formulários e Gestor de Atributos, por meio dos quais é definido e configurado todo o sistema a ser gerado. Estes gestores podem ser considerados como ferramentas de configuração dos *features* do *framework*. Após finalizar a definição do sistema a ser gerado, o Engenheiro de Aplicação utiliza o módulo Gerador de Aplicação para gerar o sistema instanciado. Assim, o usuário final utiliza o produto de software resultante, que é formado pelo código e banco de dados.

O módulo Gerador de Aplicações é responsável por fazer o mapeamento automático de uma especificação instanciada no módulo gestor do Fênix, recuperando os dados gravados em banco de dados e convertendo-os em arquivos no formato XML, de acordo com o padrão reconhecido pelo *framework* Titan, que irá gerar o código final da aplicação em linguagem PHP e, também, efetuar a criação e execução de scripts em linguagem SQL, para criação do banco de dados da aplicação final. Para tal funcionalidade, o Gerador de Aplicação é organizado em quatro módulos, conforme Figura 4.3:

(1) *WizardGuia*: configurador de interface responsável pela interação com o cliente durante o processo de criação de uma *WebApp*;

(2) *TransformerGer*: módulo gerador de código, responsável por identificar as informações de domínio instanciadas no Fênix e mapeá-las para o formato e padrão definido pelo Titan;

(3) *ScriptGer*: responsável por “montar” e ajustar os *scripts* SQL para geração do banco de dados da aplicação;

(4) *ApplicationSync*: responsável pela sincronização do código produzido pelo Titan com o banco de dados e o documento XSL, resultando na *Webapp* gerada e instalada.

Na Figura 4.3 é ilustrada a arquitetura do gerador de aplicações com seus respectivos módulos, artefatos (documentos) produzidos e iteração com o *framework* Titan para a geração da aplicação final. As seções seguintes descrevem os quatro elementos básicos responsáveis pelo processo de geração do gerador: *WizardGuia*, *TransformerGer, ScriptGer* e *ApplicationSync*.

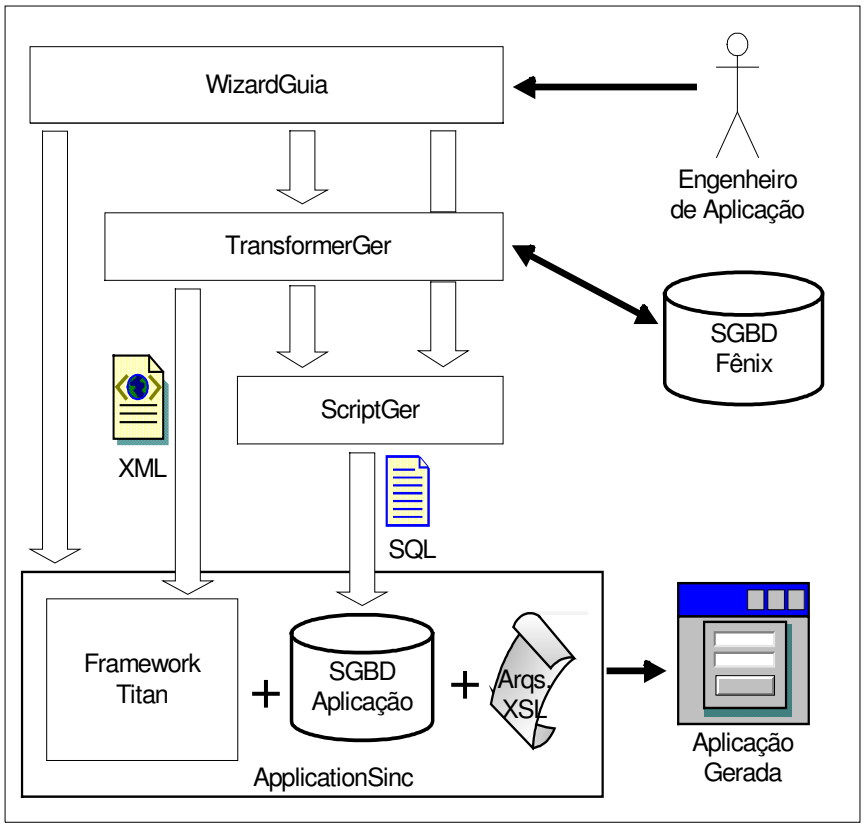


Figura 4.3: Arquitetura interna do Gerador de Aplicações.

4.3.1. Módulo *WizardGuia*

O *WizardGuia* é o módulo responsável pela interface de comunicação (formulários gráficos de comunicação com o usuário - chamados de *wizard*). Esses formulários são necessários para guiar o Engenheiro de Aplicação durante o processo de geração de uma aplicação específica, solicitando informações relevantes ao contexto, que podem ser visualizadas na Figura 4.4.

O *wizard* foi concebido de forma a solicitar o menor número de informações possível do Engenheiro de Aplicação, uma vez que o intuito é automatizar todo o processo de geração. Dessa forma, este módulo comunica-se com o Engenheiro de Aplicação para receber dados e, também com os demais módulos (*TransformerGer, ScriptGer* e A*pplicationSync*) para oferecer dados.

A interface do *Wizard* é composta por duas janelas (formulários) principais. O primeiro formulário, ilustrado na Figura 4.4, tem a responsabilidade de receber

informações gerais para geração e configuração da aplicação final, tais como diretório de instalação da aplicação na máquina cliente, endereço *Web* (*url* para acesso via *browser* de navegação *Web*) e nome da figura a ser usada como logotipo principal da aplicação.

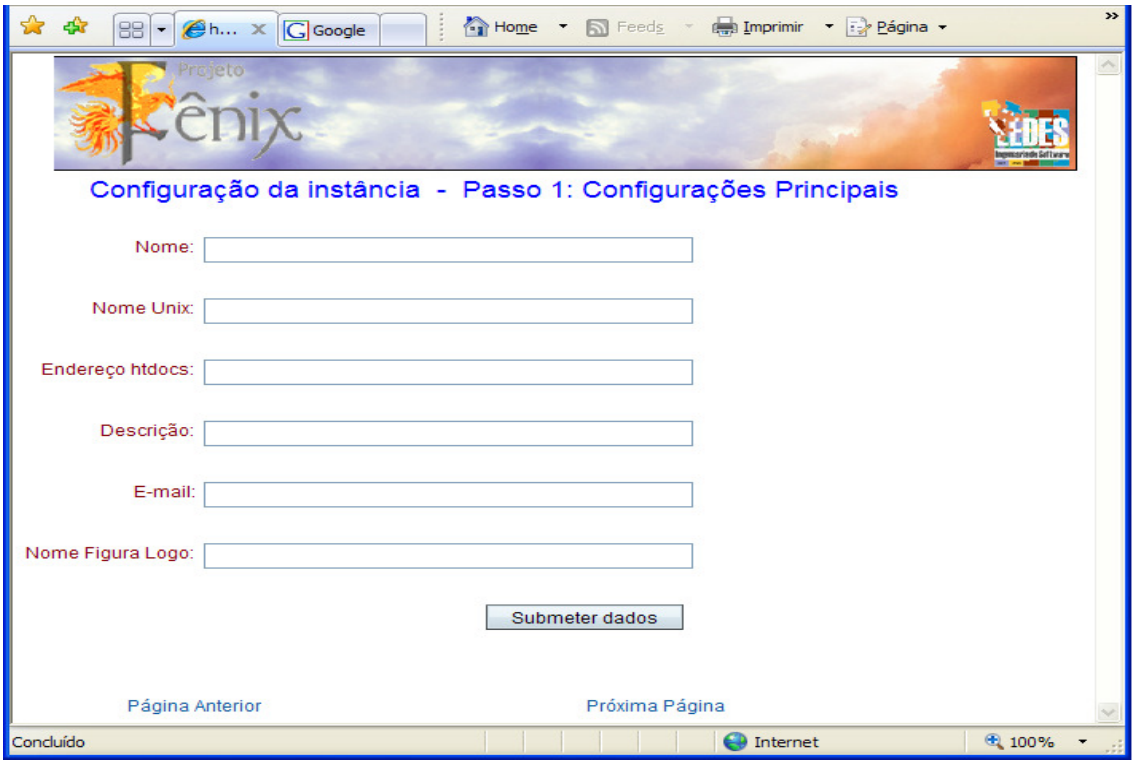


Figura 4.4: Formulário do gerador: informações gerais

No segundo formulário, o Engenheiro de Aplicação fornece informações de um SGBD específico para que o gerador possa criar as entidades da base de dados da instância gerada. Tais informações compreendem: endereço do servidor de banco de dados, nome do banco de dados, *login* e senha do usuário do banco.

Esses dados são específicos para cada aplicação a ser gerada dentro do domínio proposto. Para a implementação dessa interface e funcionalidades deste módulo foi utilizada a linguagem Java/J2EE/JSP.

Além de capturar informações e fornecê-las a outros módulos, o *WizardGuia* desempenha outra função importante: criar e montar, na máquina cliente, a estrutura de diretórios (pastas) responsáveis pelo armazenamento dos arquivos e documentos gerados para a aplicação instanciada.

Após a criação da árvore de diretórios, é realizada remotamente a cópia de todos os arquivos que formam a estrutura base do *framework* Titan. Estes arquivos são incorporados aos artefatos da aplicação instanciada e encontram-se armazenados em um repositório público do Titan, mantido pelo LEDES. Nas Figuras 4.5 e 4.6 têm-se, respectivamente, o exemplo de um trecho de código em linguagem Java que implementa cópia remota dos arquivos e o processo de criação das pastas.

```
//pasta /configure
pasta.download(endereco+"modelo/configure/"+".htaccess",pastas[w]+nome+"/configure/.htaccess");
pasta.download(endereco+"modelo/configure/"+"archive.xml",pastas[w]+nome+"/configure/archive.xml");
pasta.download(endereco+"modelo/configure/"+"business.xml",pastas[w]+nome+"/configure/business.xml");
pasta.download(endereco+"modelo/configure/"+"mail.xml",pastas[w]+nome+"/configure/mail.xml");
pasta.download(endereco+"modelo/configure/"+"security.xml",pastas[w]+nome+"/configure/security.xml");
pasta.download(endereco+"modelo/configure/"+"update.xml",pastas[w]+nome+"/configure/update.xml");
                    .                   .
                    .                   .
                    .                   .
public static void download(String address, String localFileName) {
    OutputStream out = null;  URLConnection conn = null; InputStream  in = null;
    try {URL url = new URL(address);
    out = new BufferedOutputStream(new FileOutputStream(localFileName));
    conn = url.openConnection();
    in = conn.getInputStream();
    byte[] buffer = new byte[1024]; int numRead;
    long numWritten = 0;
    while ((numRead = in.read(buffer)) != -1) {
        out.write(buffer, 0, numRead);
        numWritten += numRead;
    }
    System.out.println(localFileName + "\t" + numWritten);
    } catch (Exception exception) {
        exception.printStackTrace();
    } finally {try {if (in != null) {in.close();}
    if (out != null) {out.close();}
    } catch (IOException ioe) {}
    }
}

public static void download(String address) {
    int lastSlashIndex = address.lastIndexOf('/');
    if (lastSlashIndex >= 0 &&
            lastSlashIndex < address.length() - 1) {
        download(address, address.substring(lastSlashIndex + 1));
    } else {
        System.err.println("Não conseguiu encontrar arquivo local para " + address);
    }
}
```

Figura 4.5: Parte do código em Java para transferência remota de arquivos.

```
pastas[w] = (String)campoEndhtdocs.getText();
pasta.criar(pastas[w]+nome);
pasta.criar(pastas[w]+nome+"/cache");
pasta.criar(pastas[w]+nome+"/configure");
pasta.criar(pastas[w]+nome+"/file");
pasta.criar(pastas[w]+nome+"/image");
pasta.criar(pastas[w]+nome+"/section");
pasta.criar(pastas[w]+nome+"/section/net.ledes.access");
pasta.criar(pastas[w]+nome+"/section/net.ledes.archive");
pasta.criar(pastas[w]+nome+"/section/net.ledes.home");
pasta.criar(pastas[w]+nome+"/section/net.ledes.manager");
                .                   .
                .                   .
                .                   .
public static void criar(String nome) {
    try{
        boolean success = (new File(nome)).mkdir();
    }catch (Exception e){
    }
}
```

Figura 4.6: Parte do código em Java para criação de diretórios.

4.3.2. Módulo *TransformerGer*

Todo o comportamento da aplicação a ser instanciada encontra-se devidamente modelada e armazenada no Fênix por meio de um conjunto de módulos, estados e visões, conforme ilustrado na Figura 4.7. Cada módulo representa os diferentes tipos de projetos que deverão ser gerenciados pela aplicação gerada, por exemplo, propostas de projetos de pesquisa submetidos à avaliação de uma agência de fomento.

Figura 4.7: Comportamento de ações no Fênix. (adaptação de CARROMEU, 2007)

Cada módulo possui um conjunto finito de estados, que representam o fluxo de execução das tarefas de um módulo (*workflow* da vida do módulo). Todo o trâmite, desde a submissão da proposta, sua avaliação, seu acompanhamento técnico e administrativo são gerenciados por meio de uma série de passos que constituem o *workflow* de proposta. Cada estado do *workflow* de um Módulo agrupa um conjunto de "Visões" do sistema e cada Visão, por sua vez, constitui um relacionamento entre um Grupo de Usuários e um Formulário, ou seja, representa as permissões que cada Grupo de Usuários detêm sobre um determinado Formulário naquele estado.

O *framework* Titan é responsável por instanciar uma aplicação com base nestas informações coletadas pelo Fênix. Basicamente, o *framework* Titan especifica uma arquitetura que recebe como entrada uma linguagem de marcação (XML) e a transforma, em tempo de execução, em um gerenciador de conteúdo. Para tanto é necessário que exista uma coleção de arquivos XML de configuração e um banco de dados condizente com estes arquivos. A geração destes arquivos da base de dados e de configuração que serão a entrada do *framework* é realizada pelo módulo *TransformerGer*.

O Titan é um *framework* genérico no domínio de gerenciamento de conteúdo e, portanto, faz-se necessário um mapeamento da estrutura lógica da aplicação que o usuário definiu na ferramenta Fênix para a estrutura lógica real que é suportada pelo *framework*. O Titan trabalha com padrões como grupos de usuários, sessões e ações, enquanto o Fênix possui uma arquitetura, conforme já foi visto, baseada em formulários (subdivididos em passos, grupos e campos), módulos, atributos e grupos de usuários.

No Titan, os elementos base são as Sessões. Cada sessão é constituída por um conjunto de Ações e, para cada ação é associado um conjunto de Formulários. Cada formulário possui um conjunto de Campos e cada campo está relacionado a um atributo específico. Esse comportamento é ilustrado na Figura 4.8.

Figura 4.8: Comportamento de ações no Titan (adaptação de CARROMEU, 2007)

O gerador de código do Fênix, implementado no módulo *TransformerGer*, tem a responsabilidade de recuperar as informações armazenadas no Fênix no formato descrito anteriormente e transformá-las, produzindo arquivos XML, incorporando os padrões de domínio da aplicação a ser produzida que o Titan utiliza para gerar páginas *Web*. Estas regras de negócio são mapeadas por meio da linguagem XML.

Dessa forma, a arquitetura do módulo *TransformerGer,* responsável por transformar a linguagem de alto nível definido pelo sistema de gestão do Fênix para os padrões do *framework* Titan, foi organizado em três subcomponentes: FênixDefault, FênixModule e FênixUser.

• O subcomponente FênixDefault é responsável por exibir os módulos que um usuário pode executar da *WebApp* final, mostrando os módulos e suas respectivas descrições;

• O subcomponente FênixModule tem o propósito de gerar cada um dos módulos configurados no Fênix juntamente com seu respectivo *workflow*, obedecendo as regras e o formato definidos pelo Titan, para gerar a aplicação final. Esse esquema de conversão pode ser verificado na Figura 4.9.

• O subcomponente FênixUser é responsável por fornecer um módulo para gerenciar os usuários da aplicação final, um módulo para atualizar o cadastro do usuário e fornecer os devidos formulários de cadastros para os usuários públicos, semi-públicos e privados. Esses módulos citados foram definidos pelo subcomponente FênixUser, ou seja, não são configuráveis no *framework*.

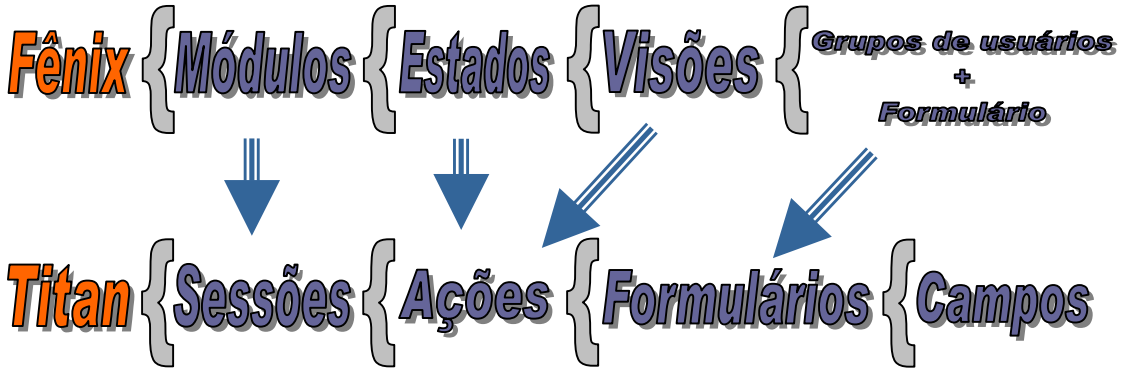


Figura 4.9: Transformações dos modelos do Fênix para o Titan.

Dessa maneira, o subcomponente FênixDefault corresponde a uma sessão de uma aplicação qualquer gerada pelo *framework* Titan, responsável por exibir os módulos e suas respectivas descrições da aplicação final. Este subcomponente receberá documento XML como apresentado na Figura 4.10, que o informa por meio de diretivas, o número de módulos (NUMBER_OF_MODULES), os nomes dos módulos (MODULE_i_NAME), as descrições dos módulos (MODULE_i_DESCRIPTION) e os identificadores dos arquivos XML de configuração dos módulos (MODULE_i_ID), onde *i* representa o índice do módulo, que tem o numero *0* como índice inicial.

O subcomponente FênixUsers é responsável por gerar os módulos adicionais de gerência de usuários e atualização de cadastros. Além destes módulos este subcomponente fornece os formulários de cadastros de usuários. No contexto da Aplicação Fênix é definido que um usuário sempre pertence a um único grupo, e cada grupo é definido por um nome e domínio, onde o domínio assume apenas três valores:público, semi-público e privado.

```
<section>
  <component>
      <name>fenixDefault</name>
      <version>1.0</version>
      <path>titan/components/fenixDefault/</path>
  </component>

  <directive name="NUMBER_OF_MODULES">2</directive>
  <directive name="MODULE_0_NAME">Cadrasto de aluno</directive>
  <directive name="MODULE_1_NAME">Propostas de Projetos</directive>

  <directive name="MODULE_0_DESCRIPTION">Desc. mód. 1</directive>
  <directive name="MODULE_1_DESCRIPTION">Desc. mód. 2</directive>

  <directive name="MODULE_0_ID">fenix_module_52</directive>
  <directive name="MODULE_1_ID">fenix_module_11</directive>

  <action default="true" name="default" />
</section>
```

Figura 4.10: Modelo de XML FênixDefault.

O módulo de gerência de usuários da aplicação é disponibilizado apenas ao administrador da *WebApp* gerada. Este módulo fornece ao administrador as seguintes funcionalidades: habilitar e desabilitar grupos, usuários e permissões

dos grupos. Também oferece a opção de criar usuários de domínio privado, sendo equivalente a uma Sessão de uma aplicação gerada pelo *framework* Titan.

O subcomponente FênixModule tem como funcionalidade gerar um módulo de uma aplicação no contexto do Sistema Fênix. Um módulo, no contexto do Fênix, pode ser descrito como um *workflow*, no qual cada conjunto de atividades é denominada Estado. Um estado é definido como um conjunto de visões, e cada visão é definida como um conjunto de permissões que um determinado grupo de usuários tem sobre um determinado formulário. As possíveis permissões que um grupo de usuários pode ter sobre um formulário são: editar, incluir, apagar e visualizar. Portanto, o subcomponente FênixModule cria um documento XML contendo as configurações de um módulo para gerar uma Sessão de uma aplicação gerada pelo *framework* correspondente a esse módulo. Esta Sessão correspondente a um módulo, tem como ação padrão a exibição dos estados do módulo, sendo que cada estado exibe suas Visões. Estas informações são passadas por meio de diretivas como: NUMBER_OF_STATES, informa o número de estados de um módulo; STATE_i_NAME, nome do estado i; STATE_i_DESCRIPTION, uma breve descrição sobre a função do estado i; STATE_i_NUMBER_OF_VISIONS, informa o numero de visões que o estado i possui; STATE_i_VISION_j_NAME, informa o nome da visão j do estado i; STATE_i_VISION_j_ACTION, informa qual ação será executada na visão j do estado i, onde i representa o índice do estado e j o índice da visão, e eles tem o número 0 como índice inicial. Um exemplo de código XML gerado é mostrado na Figura 4.11.

```
<directive name="NUMBER_OF_STATES">2</directive>
<directive name="STATE_0_NAME">Estado 1</directive>
<directive name="STATE_1_NAME">Estado 2</directive>
<directive name="STATE_0_DESCRIPTION">Cad. alunos</directive>
<directive name="STATE_1_DESCRIPTION">Cad. professor</directive>
<directive name="STATE_0_NUMBER_OF_VISIONS">1</directive>
<directive name="STATE_0_VISION_0_NAME">visão 1</directive>
<directive name="STATE_0_VISION_0_ACTION">list_vision_52</directive>
<directive name="STATE_1_NUMBER_OF_VISIONS">1</directive>
<directive name="STATE_1_VISION_0_NAME">visão 1</directive>
<directive name="STATE_1_VISION_0_ACTION">list_vision_53</directive>
```

Figura 4.11: Modelo de XML FênixModule.

O Fênix define os seguintes atributos: *bool, cep, cnpj, coin, cpf, date, enum, file, float, int, city, state, password, string, text, time, list* e *subform*. O Titan suporta quase todos esses atributos, exceto os atributos *list* e *subform* e algumas extensões do tipo *select*, cujo suporte ainda será implementado. Esses três atributos serão usados para simular o relacionamento entre as tabelas do banco de dados da aplicação gerada. Para cada módulo da aplicação deverá existir uma tabela no banco de dados relacional que será a fonte de dados do módulo, e é denominada tabela principal do módulo. Portanto, os relacionamentos são sempre entre uma tabela principal, que é a base da execução de um módulo, e uma outra tabela, que pode ser uma tabela principal de outro módulo ou uma tabela secundária. Uma tabela secundária é definida como uma tabela que mantém um relacionamento de dependência com uma tabela principal. O processo usado para mapear e criar as tabelas será explicado na seção seguinte.

Optou-se por utilizar a linguagem XML pelas suas características, que facilitam o processo de conversão entre tipos de dados diferentes e, também, por ser o formato padrão de entrada de dados requerido pelo *framework* Titan. O Titan utiliza a linguagem PHP para gerar o código da aplicação instanciada. Para gerar código em outra linguagem, é necessário implementar esta funcionalidade no *framework* Titan.

4.3.3. Módulo *ScriptGer*

O módulo *ScriptGer* tem como responsabilidade “montar” e ajustar os *scripts* SQL para geração do banco de dados da aplicação. O SGBD é o mecanismo responsável pela persistência e controle dos dados, devendo ser capaz de disponibilizar os dados durante todo o tempo de execução do sistema. Na arquitetura proposta, o banco de dados é implementado por meio de *scripts* em linguagem SQL, que são criados dinamicamente pelo gerador de código, obedecendo regras de consistência previamente estabelecidas e mapeadas para um SGBD relacional.

Basicamente, o *ScriptGer* desempenha três tarefas distintas e consecutivas:

1) Montagem dos *scripts* em linguagem SQL para criação das tabelas específicas da aplicação instanciada e armazenada no Fênix. Este processo ocorre por meio de leitura nos arquivos XML criados no módulo *TransformerGer*;
2) Leitura dos *scripts* SQL e criação das tabelas físicas e seus devidos relacionamentos e dependências no Banco de Dados; e
3) Povoamento de algumas tabelas com dados necessários para iniciar a execução da aplicação final.

Com as informações armazenadas na base de dados do Fênix, o Gerador de Aplicações, por meio do módulo *TransformerGer*, obtém as informações sobre a aplicação a ser gerada e converte-as em documentos padrões em formato XML (conforme descrito na seção anterior). Assim, para cada padrão usado na aplicação, é feita análise de quais tabelas devem ser criadas com seus respectivos atributos e chaves. Cada arquivo XML é analisado individualmente sendo gerados três arquivos distintos (seguindo as regras SQL determinadas pelo PostgreeSQL), porém relacionados: *db.sql, pk.sql* e *fk.sql.* Cada um destes arquivos é montado com base em modelos que servem como base e seguem o padrão do banco de dado PostgreeSQL para gerar as tabelas. Analisando-se alguns *scripts* existentes para tal finalidade, partes fixas e variáveis foram identificadas. A Figura 4.13 apresenta um conjunto de *scripts* de criação de tabelas usando SQL, sendo que os itens circulados representam as partes fixas encontradas nos *scripts* analisados. A Figura 4.12 apresenta o gabarito (modelo) definindo esses *scripts*, as partes fixas estão definidas de acordo com a identificação na Figura 4.13 e as partes variáveis são colocadas entre *tags*. O gabarito definido na Figura 4.12 é utilizado para a criação do arquivo *db.sql*.

```
CREATE TABLE <NOME_DA_TABELA> (
    <ATRIBUTOS> <TIPOS>(<TAMANHOS>) NULL);
```

Figura 4.12: Gabarito definido para os *scripts* de criação de tabelas.

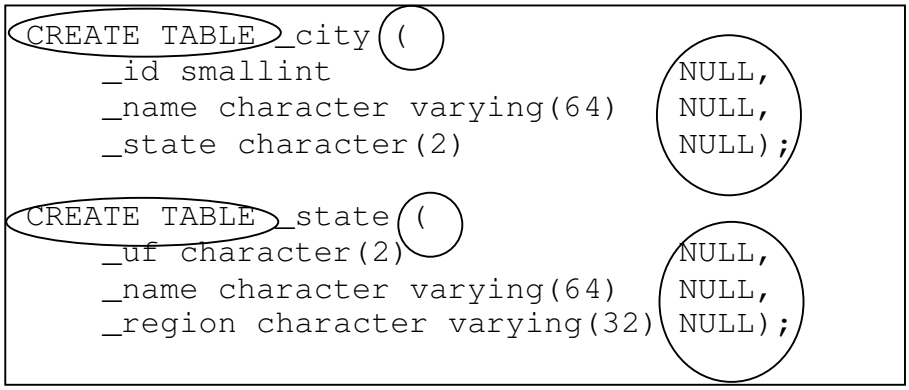


Figura 4.13: Conjunto de *scripts* de criação de tabelas usando SQL.

O artefato *pk.sql* altera a estrutura das tabelas definidas no artefato *db.sql* criando as suas chaves primárias (*primay key*). Para a sua elaboração, a *tag* <*atributos*> é analisada e, uma vez que for definido como chave primária, a estrutura do campo é alterada para não permitir valor nulo e uma chave primária com prefixo *"PK_"*, seguido do nome da tabela atual é criada. A Figura 4.14 apresenta o gabarito de definição das chaves primárias.

```
// para cada atributo definido como pk
ALTER TABLE <NOME_DA_TABELA> (
   MODIFY <VALOR DA TAG ATRIBUTO PK> <TIPO PK>(<TAMANHO PK>)
   NOT NULL);

ALTER TABLE <NOME_DA_TABELA> (
   ADD (CONSTRAINT PK_<NOME DA TABELA>
      PRIMARY KEY(<VALOR DA TAG ATRIBUTO PK>));
```

Figura 4.14: Gabarito de definição das chaves primárias.

Finalmente, o artefato *fk.sql* altera a estrutura das tabelas definidas no artefato *db.sql* criando as chaves estrangeiras (*foreing key*) definindo os relacionamentos entre as tabelas. Para a sua elaboração são analisadas as *tags* que referenciam as associações no documento XML. Sempre que existir uma associação entre formulários, por exemplo, um relacionamento deve ser estabelecido entre as tabelas da base de dados. Para isso é adicionada uma chave estrangeira com o prefixo *"FK_"* seguido do nome da tabela que está sendo atualmente analisada pelo gerador e o nome da tabela associada, com seu

respectivo atributo de associação. A Figura 4.15 apresenta o gabarito de definição das chaves estrangeiras.

```
// para cada associação encontrada
ALTER TABLE <NOME_DA_TABELA ATUAL> (
 ADD (CONSTRAINT FK_<NOME DA TABELA ATUAL>_<NOME DA TABELA
      ASSOCIADA>
 FOREIGN KEY(FK_<NOME DA TABELA ASSOCIADA>)
 REFERENCES <NOME DA TABELA ASSOCIADA>
 (<VALOR DA TAG ATRIBUTO PK DA TABELA ATUAL>);
```

Figura 4.15: Gabarito de definição das chaves estrangeiras.

A Figura 4.16 exemplifica a construção dos artefatos SQL para construção de tabelas. Os caracteres em negrito da Figura 4.13 são as partes fixas do gabarito, embutidos no código Java, como mostra a Figura 4.16 (a). Dessa forma, o gerador deve completar as informações como *NOME_DA_TABELA, ATRIBUTOS, TIPOS* e *TAMANHOS* a partir das definições XML geradas no módulo *TransformerGer*. O artefato gerado é então mostrado na Figura 4.16 (b).

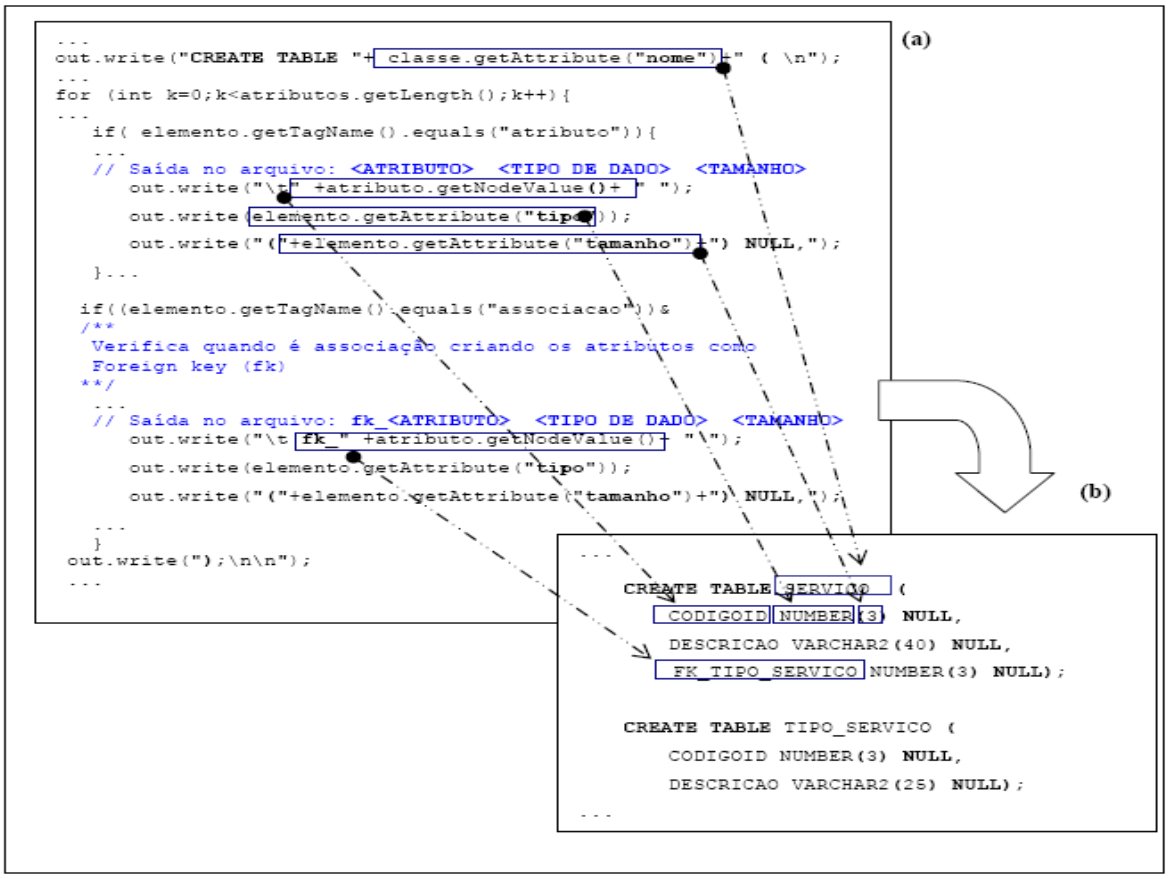


Figura 4.16: Esquema de construção dos *scripts* SQL de criação de tabelas.

Após finalizar o processo de mapeamento e criação dos *scripts* SQL, o módulo *ScriptGer* executa-os para efetivar a criação das tabelas físicas e seus devidos relacionamentos e dependências no Banco de Dados e, finalmente faz o povoamento de algumas tabelas com dados essenciais, como criar um usuário administrador e atribuir suas devidas responsabilidades. Também são povoadas as tabelas de estados e cidades entre outras. A implementação dessas tarefas também é feita com código em Java e um exemplo do código que executa um script para criação de tabelas no banco é visualizado na Figura 4.17.

```
public static void criarbanco(String nomeBanco, String usuario, String senha,String servidor)
    throws ClassNotFoundException {
    Class.forName("org.postgresql.Driver");
    Connection con = null;
    try {
        con=DriverManager.getConnection("jdbc:postgresql://"+servidor+"/"+nomeBanco,usuario,senha);
    } catch (SQLException ex) {ex.printStackTrace();}
    try {Statement s=con.createStatement();
    try {
        BufferedReader in = new BufferedReader(new FileReader("db.sql"));
        String str;
        while ((str = in.readLine()) != null) {
            s.execute(str);}
        in.close();
    } catch (IOException e) {
    }
    s.close();
    con.close();
    }catch (Exception e) {}

}
```

Figura 4.17: Exemplo de código para execução de *scripts* SQL.

No Fênix é possível a criação de *scripts* SQL para a geração de qualquer SGBD suportado pelo *framework* Hibernate. No entanto, atualmente o *framework* Titan só suporta geração de código para aplicações que utilizem os bancos de dados PostgreSQL, razão pela qual em um primeiro momento restringe a escolha ao banco de dados mencionado. Futuramente pretende-se estender essa funcionalidade do Fênix também para o Titan, oferecendo assim, um numero maior de opções para escolha do SGBD da aplicação gerada.

4.3.4. Módulo *ApplicationSync*

Finalmente, o módulo *AplicationSync* contém as configurações da aplicação como: os arquivos de configuração geral da aplicação, configurações das seções, configurações do mapeamento do banco de dados e configuração do sistema de controle de usuários. O documento em formato XSL, especificado nas Figuras 4.2 e 4.3, é um arquivo de configuração da interface da aplicação gerada. Neste arquivo ficam localizados todos os "*skins"* da aplicação ("*Banners"*, imagens de fundo de tela, etc). Este documento é produzido pelo Engenheiro de Aplicação antes de iniciar o processo de geração da aplicação.

4.4. Tecnologias Adotadas

Como o Gerador de Aplicação é um dos módulos da ferramenta Fênix, as tecnologias adotadas têm como base o paradigma de software livre:

- EJB3 (SUN, 2006): suporta a arquitetura baseada em componentes e é um aprimoramento da conhecida tecnologia Enterprise Java Beans (EJB);
- XML (*eXtensible Markup Language*) ([EISENBERG, 2002): Tecnologia utilizada no Gerador de Código para mapear as regras de negócios instanciadas no Fênix e submetê-las ao Titan;
- Java/J2EE/JSP (ECKEL, 2003): A linguagem Java e seus derivados formam a base da ferramenta Fênix. Esta tecnologia foi escolhida por sua portabilidade e pelo fato de ser uma tecnologia que segue o paradigma do software livre, ou seja, sua licença é gratuita e possui código aberto;
- *Framework* Hibernate (BAUER, KING, 2005): Responsável pela camada de persistência do Fênix. Foi escolhido pelo fato de tornar a ferramenta independente de SGBD, ou seja, apesar de ter sido adotado o PostgreSQL o Fênix é totalmente portável, sem necessidade de implementação para outro SGBD;
- PostgreSQL (STINSON, 2001): Sistema de Gerência de Banco de Dados (SGBD) utilizado no desenvolvimento e como repositório para a base de dados da aplicação a ser gerada. Foi escolhido por seguir o paradigma de software livre e por se enquadrar como banco de dados robusto e estável; e

- PHP (CONVERSE, PARK, 2003): A linguagem PHP está sendo utilizada para implementação de uma instância do *framework* Titan. Por ser livre e de código aberto foi escolhida para mostrar que o Fênix e o Titan, apesar de implementados em diferentes linguagens de programação, poderão interagir normalmente utilizando apenas a linguagem genérica para geração de aplicações Web no domínio especificado.

4.5. Considerações Finais

Neste capítulo foi apresentada a arquitetura do gerador automático de *WebApps* baseado no *framework* Titan e integrado à ferramenta Fênix, que auxilia no processo de desenvolvimento de aplicações no domínio de Sistemas Web de Apoio à Gestão de Fomento de Projetos. A arquitetura do gerador é baseada em quatro módulos principais, sendo o primeiro composto por um *Wizard* responsável pela interação com o Engenheiro de Aplicação durante o processo de geração da aplicação. O segundo módulo faz o mapeamento dos dados armazenados em banco de dados no Fênix para uma representação especificada pelo *framework* Titan. Já o terceiro módulo é responsável pela criação dos *scripts* em linguagem SQL e geração do banco de dados da aplicação. Finalmente, o quarto e último módulo, destinado a tratar questões de sincronização e configurações de documentos, resultando na aplicação Web gerada e instalada. No próximo capítulo será descrito o processo de geração de uma aplicação Web, no domínio proposto, a partir da ferramenta Fênix, bem como a arquitetura geral da WebApp instanciada.

Capítulo 5
Processo de Geração: Um Estudo de Caso

5.1. Considerações Iniciais

Neste capítulo é apresentado um exemplo do processo de geração de uma aplicação Web no domínio proposto, a partir da ferramenta Fênix, bem como a arquitetura geral da *WebApp* instanciada. Para facilitar o entendimento do comportamento do gerador de aplicação, será utilizado como estudo de caso o Sistema de Informação em Pesquisa Universitária (SIPES), implementado segundo os padrões definidos no Fênix.

O capítulo está organizado da seguinte forma: na Seção 5.2 é apresentada uma visão geral da aplicação instanciada. A Seção 5.3 exemplifica o processo de geração da aplicação Web específica; a Seção 5.4 descreve a arquitetura geral da aplicação final e as considerações finais sobre este capítulo são realizadas na Seção 5.5.

5.2. Descrição da Aplicação

Para efetuar a validação do gerador de aplicações, foi projetada uma aplicação simples, denominada SIPES, modelada segundo os padrões definidos pela ferramenta Fênix e gerada automaticamente por meio de sua ferramenta geradora, objeto deste trabalho, em conjunto com o *framework* Titan.

A *WebApp* tem como objetivo apresentar um cadastro e controle de projetos de pesquisa e de iniciação científica realizados na Pró-Reitoria de Pesquisa e Pós-Graduação da UFMS, orientando os alunos e demais membros da comunidade acadêmica da universidade.

Dessa forma, o sistema gerado deve conter, entre outras funcionalidades, as seguintes:

- Modalidades de Pesquisa: menu responsável por gerenciar os dois módulos principais do sistema: Projetos de Pesquisa e Iniciação Científica;
- Pessoas: nesta opção são gerenciados os usuários do sistema. É dividido por Gestores e Pesquisadores (alunos e professores);
- Instituição: módulo para cadastro e manutenção de dados da instituição;
- Editais: cadastro de editais de órgãos de fomento para submissão de projetos de pesquisa;
- Áreas de Conhecimento: cadastro de áreas de conhecimento, as quais são utilizadas para delimitar e controlar o escopo de cada projeto de pesquisa ou de iniciação científica;
- Configuração do Portal: área para configuração de duas sessões: Notícias e Banco de Arquivos; e
- Controle de Acesso: módulo administrador, possibilitando a gerência e administração do sistema por meio de operações como criação de grupos de trabalho e definição de permissões para grupos e usuários.

5.3. Processo de Geração de uma *WebApp*

O processo começa a partir da definição e especificação de todos os requisitos e funcionalidades da instância a ser gerada no sistema de gestão do Fênix pelo Engenheiro de Aplicação. Estas atividades consistem em definir os módulos da aplicação, seus formulários (interfaces gráficas com o usuário), conjunto de atributos necessários a cada interface e os grupos de usuários da aplicação.

A partir da descrição das funcionalidades do sistema, descritas na seção anterior, são extraídos os requisitos e identificados os casos de uso da aplicação a ser instanciada. A Figura 5.1 mostra, de forma simplificada, os principais casos de uso e sua iteração com os atores externos em um Diagrama de Casos de Uso, no qual pode-se observar a existência de dois casos de uso principais: “Gerenciar Projetos de Iniciação Científica” e “Gerenciar Projetos de Pesquisa”, que interagem diretamente com outros casos de uso importantes para a aplicação, como “Gerenciar Instituições”, “Gerenciar Áreas de Conhecimento” e “Gerenciar Editais”. O Diagrama de Casos de Uso é um diagrama mais geral e informal,

mostrando uma visão estática de casos de uso do sistema, sendo importantes, principalmente, para a organização e modelagem do comportamento do sistema.

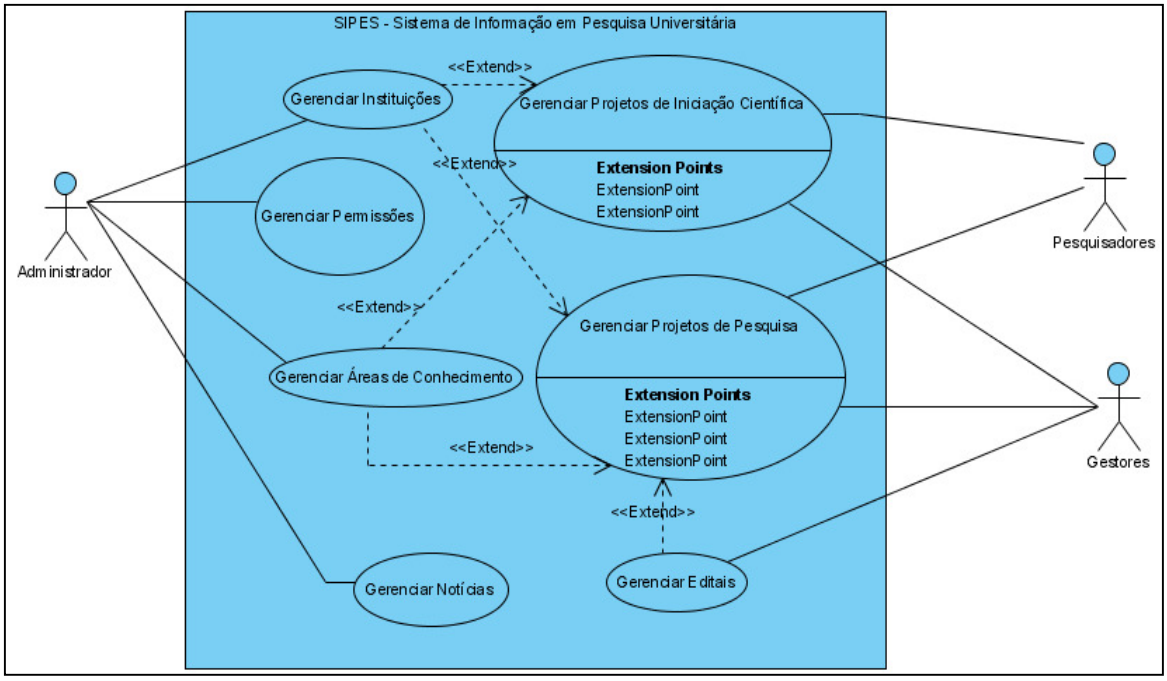


Figura 5.1: Diagrama de Casos de Uso simplificado da aplicação.

As definições do diagrama de casos de uso servem de base para elaboração do Diagrama de Classes. Na Figura 5.2 é mostrada uma representação simplificada do Diagrama de Classes da aplicação a ser instanciada. Esta representação é utilizada pelo fato de possibilitar a visualização e projeto do conjunto de interfaces (formulários) dos principais módulos do sistema, bem como seus respectivos atributos. Dessa forma, pode-se observar, pelo diagrama mostrado na Figura 5.2, a existência de classes representando possíveis formulários a serem instanciados, como por exemplo, formulários para gerenciar (inserir, alterar, consultar, atribuir permissões) usuários, projetos de pesquisa, projetos de iniciação científica, cadastro de instituições de ensino e pesquisa, editais de seleção de projetos de pesquisa e cadastro de áreas de conhecimento.

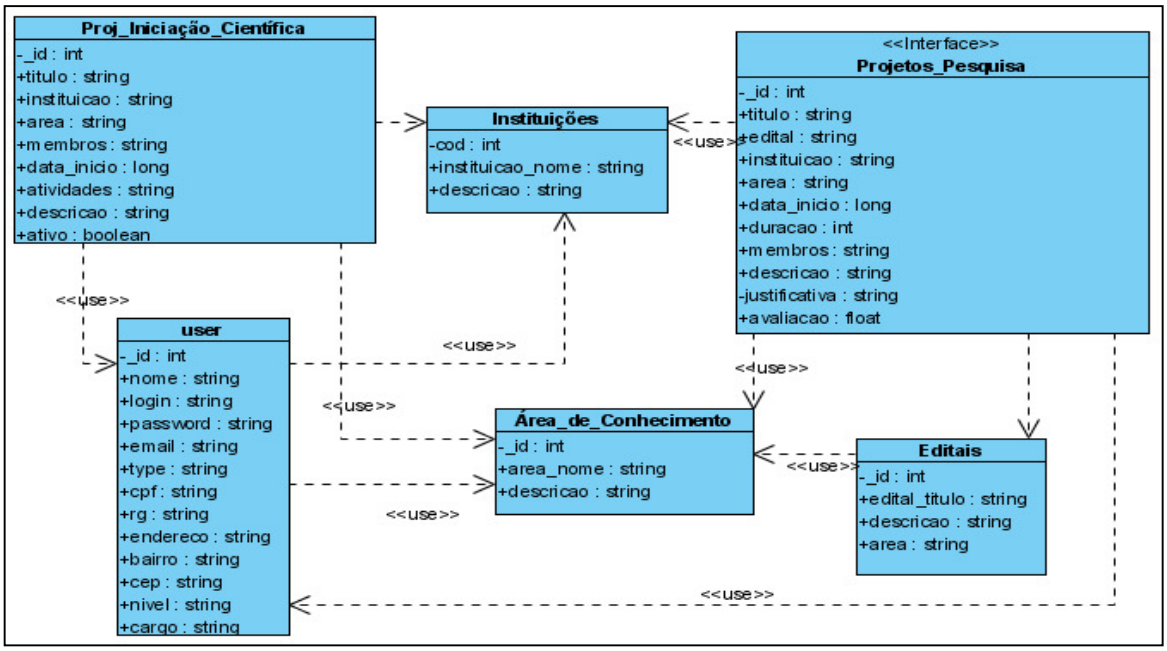


Figura 5.2: Diagrama de Classes simplificado da aplicação.

Finalizada a etapa de projeto do sistema, cabe ao Engenheiro de Aplicação a tarefa de inserir essas informações no Fênix para efetivamente gerar a aplicação final. Esses dados são coletados no Fênix por meio de um conjunto de formulários do módulo Sistema de Gestão.

O Sistema de Gestão do Fênix define toda a infra-estrutura para permitir a configuração das *features* da LPS. O primeiro passo no processo de instanciação do Fênix é a criação de uma nova aplicação. Uma aplicação dentro da ferramenta é uma futura instância do *framework* Titan, e pode ser criada por meio da interface apresentada na Figura 5.3.

Figura 5.3: Interface para gerência de aplicações no Fênix.

O segundo passo do processo de instanciação é a configuração da aplicação, que consiste em definir os módulos da aplicação, seus formulários (interfaces gráficas com o usuário), conjunto de atributos necessários a cada interface e os grupos de usuários da aplicação. Na Figura 5.4 é apresentada a interface do Fênix, que contém um menu que direciona o usuário a realizar todas essas configurações, ou seja, criar, editar e apagar novos Módulos, Grupos de Usuários, Atributos e Formulários para a aplicação a ser instanciada. Dessa forma, o sistema de Gestão do Fênix gerencia o funcionamento de quatro outros subsistemas gestores: Gestor de Usuários, Gestor de Atributos, Gestor de Formulários e Gestor de Módulos, conforme descritos no Capítulo 3.

Figura 5.4: Menu de seleção de gestores de configuração.

Dessa forma, o Engenheiro de Aplicação segue a seqüência de eventos para realizar todas as configurações necessárias para a geração da aplicação modelada. Por exemplo, após a criação dos usuários (definição dos tipos de usuários que a aplicação terá), utiliza-se o módulo “Formulários” para criar os formulários. A ação de criar um formulário resume-se, basicamente em informar o nome do mesmo e o seu tipo (se é um formulário “*main*” ou descendente de outro). A Figura 5.5 mostra um resumo dos formulários criados para o SIPES.

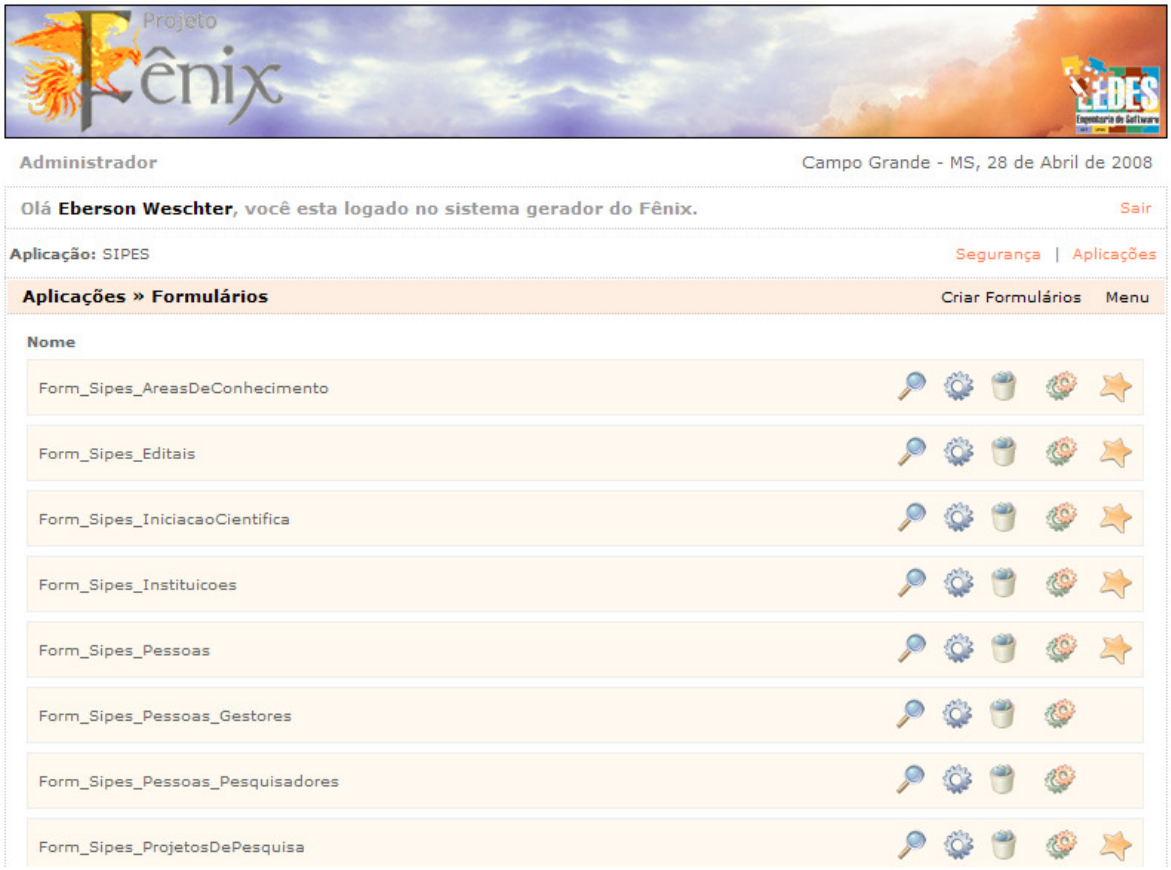


Figura 5.5: Resumo de formulários criados para o SIPES.

Uma vez definidos os formulários da aplicação, o Engenheiro segue para o próximo passo, a criação dos módulos e os respectivos formulários que fazem parte de cada módulo. Por exemplo, para o módulo “Pessoas” criado, fazem parte os formulários para cadastro e gerência de Gestores e cadastro e gerência de Pesquisadores. Para cada formulário, devem ser informados quais campos (atributos), com seus respectivos tipos, fazem parte.

Cada formulário com o “status” de “Principal”, ou seja, um formulário acessado a partir do “*Main*” (tela principal da aplicação) é mapeado em uma tabela no banco de dados relacional e cada atributo torna-se uma coluna. Na Figura 5.6 têm-se um formulário com visualização parcial de alguns campos criados e seus respectivos tipos.

Ainda no Módulo Gestor do sistema Fênix, o Engenheiro define a seqüência de passos para cada formulário, agrupados em grupos e, também,

quais campos pertencem a cada grupo dentro de um formulário especifico. Esses dados são gravados em banco de dados e recuperados posteriormente, pelo Modulo Gerador de Aplicação, objeto deste trabalho.

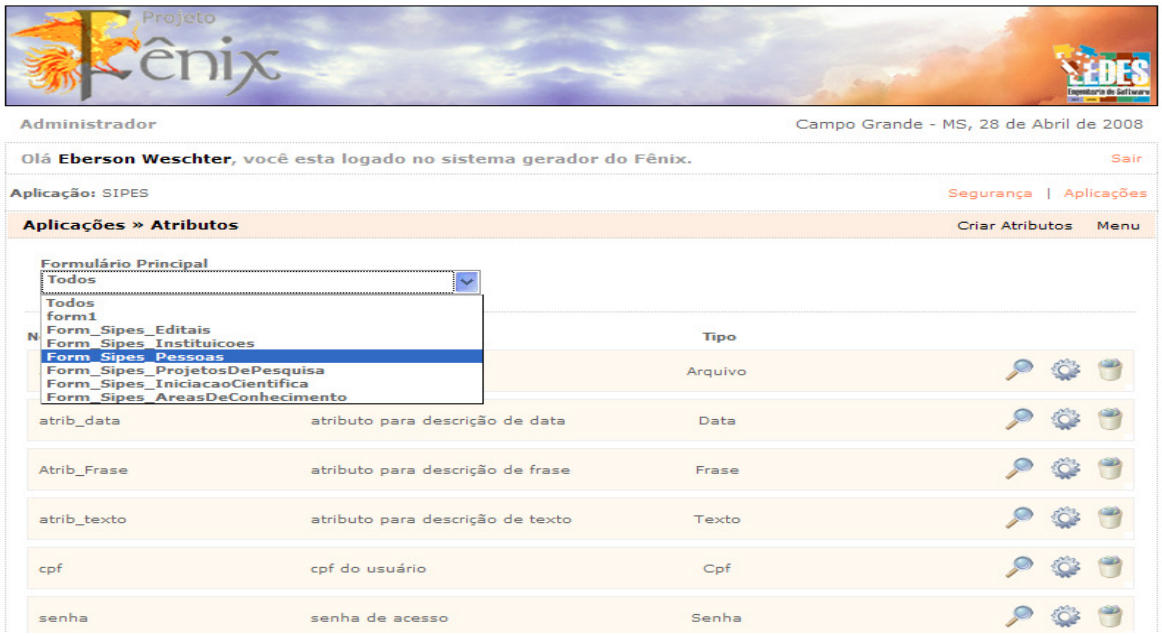


Figura 5.6: Visualização de campos criados para formulários.

Após a finalização desta etapa, o Engenheiro de Aplicação tem acesso ao módulo gerador de aplicação a partir do sistema gerador do Fênix, conforme mostrado na Figura 5.7.

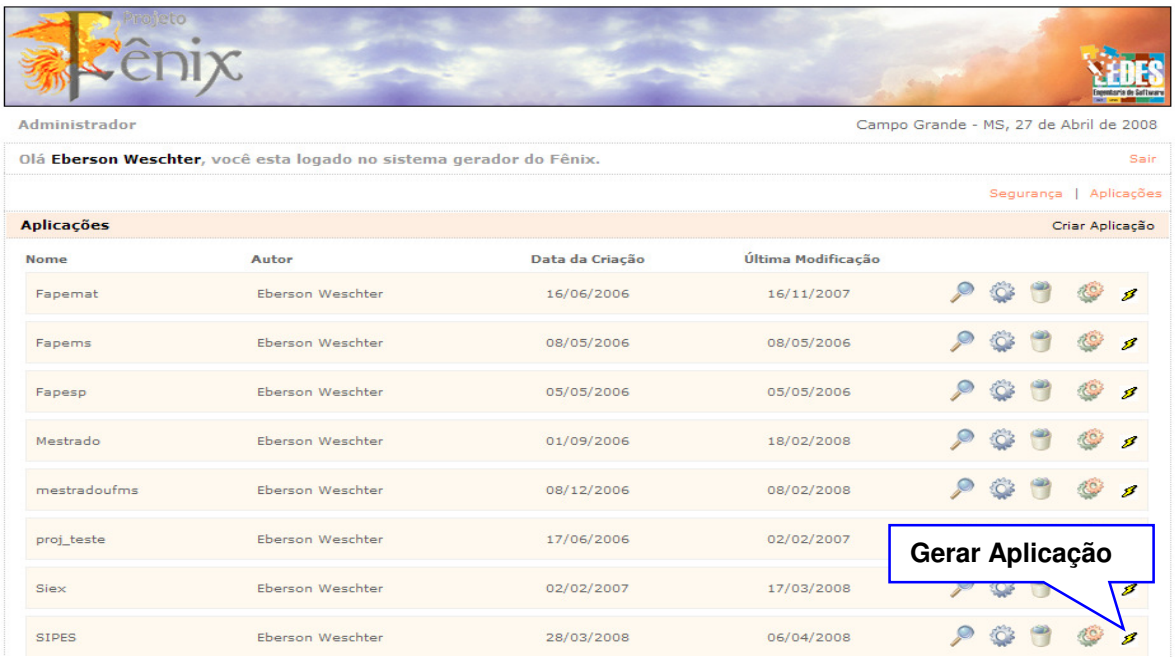


Figura 5.7: Acesso ao gerador de aplicação a partir do sistema gestor do Fênix.

O sistema gerador constitui-se, basicamente de dois formulários (telas) principais, os quais solicitam ao Engenheiro de Aplicação algumas informações necessárias para completar o processo de criação e instalação da aplicação final. No primeiro formulário são solicitadas informações gerais para instalação e configuração da aplicação final. Os dados requisitados são: nome da aplicação; nome do diretório (pasta) base a ser criado e onde serão instalados os demais arquivos da aplicação na máquina cliente (servidor da *WebApp*); endereço *htdocs* (caminho físico para localização do diretório do servidor de aplicações *Web* que serve como repositório público); descrição breve da aplicação (título ou *slogan*) e nome e localização da figura (*skin*) a ser usada como logotipo principal da aplicação (opcional).

O segundo formulário, mostrado na Figura 5.8, é destinado a tratar questões relacionadas ao banco de dados da aplicação final. Nesta etapa o Engenheiro de Aplicação deve fornecer informações de acesso a um SGBD PostgreeSQL, referentes a um usuário previamente criado no banco de dados. Tais informações compreendem: endereço do servidor de banco de dados, nome do banco de dados, *login* e senha do usuário do banco. Esses dados são necessários para criar as entidades da base de dados da instância gerada.

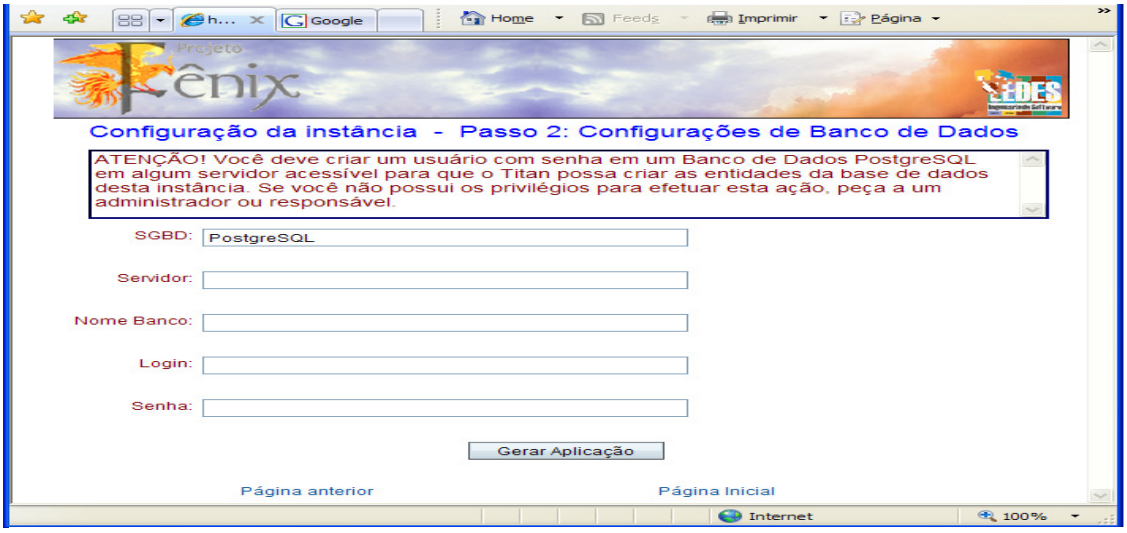


Figura 5.8: Formulário do gerador: informações do banco de dados.

Após o correto preenchimento dos dados no formulário da Figura 4.4 (referenciada no Capítulo 4), o Engenheiro de Aplicação dá início ao processo

automatizado para geração da aplicação final, por meio do botão *Gerar Aplicação*. A partir deste instante o gerador monta, na máquina cliente, a estrutura de diretórios (pastas) responsáveis pelo armazenamento dos arquivos e documentos gerados para a aplicação instanciada. Um exemplo da estrutura criada é mostrada na Figura 5.9, destacando o diretório base (principal) *"SIPES"* , a partir do qual são criados os demais.

A partir desse instante o gerador executa os módulos *TransformerGer* e *ScriptGer* (descritos no Capítulo 4), responsáveis pela montagem dos documentos XML e *scripts* para criação do banco de dados. Esses artefatos são submetidos ao *framework* Titan, que irá gerar a aplicação final.

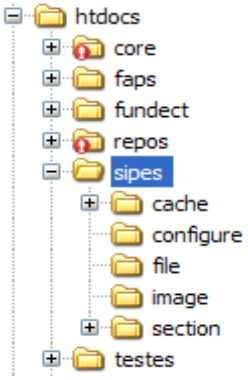


Figura 5.9: Estrutura de diretórios da aplicação instanciada.

A aplicação final gerada é constituída, basicamente pelo módulo gestor, e a *WebApp* contendo todas as regras de negócio, mostrado na Figura 5.10. O modulo gestor é utilizado pelo usuário administrador do sistema para executar algumas tarefas pertinentes, como criação de grupos de notícias e cadastro e controle de permissões de usuários gestores.

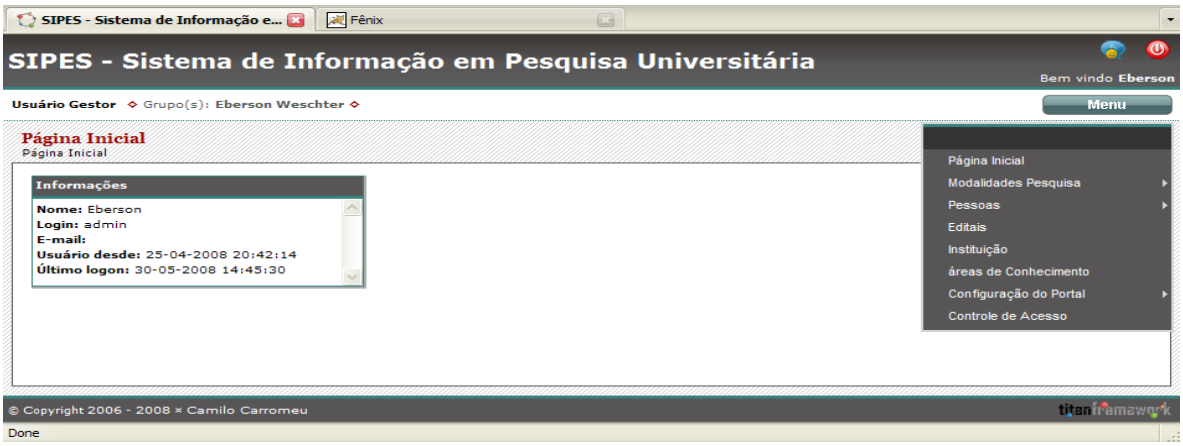


Figura 5.10: Tela principal da aplicação gerada.

Todo o comportamento da aplicação a ser instanciada, conforme já discutido nos capítulos anteriores, encontra-se devidamente modelada e armazenada no Fênix, por meio de um conjunto de módulos, estados e visões. Cada módulo representa os diferentes tipos de projetos que são gerenciados pela aplicação gerada, por exemplo, propostas de projetos de pesquisa submetidos à avaliação de uma agência de fomento. Na Figura 5.11 é apresentada a página do sistema SIPES que ilustra todos os módulos disponíveis: Projetos de Pesquisa e Iniciação Científica.

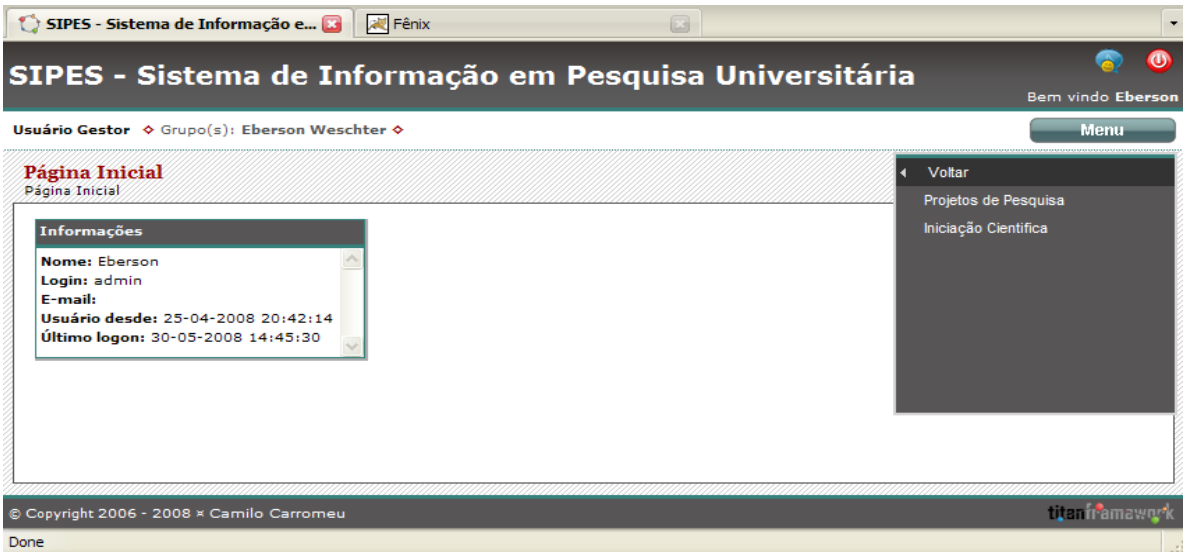


Figura 5.11: Página no sistema SIPES ilustrando acesso aos Módulos.

Cada módulo possui um conjunto finito de estados, que representam o fluxo de execução das tarefas de um módulo (*workflow* da vida do módulo). Um módulo representa os diferentes tipos de processos que são controladas pela aplicação gerada, por exemplo, propostas de projetos de pesquisa submetidos à avaliação de uma agência de fomento. Todo o trâmite, desde a submissão da proposta, sua avaliação, seu acompanhamento técnico e administrativo serão gerenciados por meio de uma série de passos que constituem o *workflow* de proposta. Cada estado do *workflow* de um Módulo agrupa um conjunto de "Visões" do sistema e cada Visão, por sua vez, constitui um relacionamento entre um Grupo de Usuários e um Formulário, ou seja, representa as permissões que cada Grupo de Usuários detêm sobre um determinado Formulário naquele

estado. Por exemplo, no SIPES uma proposta de projeto que esteja na situação (estado) “Sob avaliação de Gestores” poderá ser visualizada pelo grupo de usuários Pesquisador (o coordenador e os membros da proposta), pelo grupo Gestores (que irão editar um formulário de avaliação), e poderá ser visualizada e editada pelos Administradores (que irão duplicar algumas informações da proposta, criando uma cópia que poderá ser modificada).

Na Figura 5.12 é apresentada a página do Sistema de Informação em Pesquisa Universitária utilizado como exemplo, que ilustra um passo de um formulário do módulo Projeto de Pesquisa.

Figura 5.12: Página do SIPES ilustrando parte de uma Sessão.

Alguns dos demais passos deste módulo estão listados na Figura 5.13, que são: Descrição da Pesquisa, Justificativa da Pesquisa, Objetivos da Pesquisa, Cronograma de Atividades, Metodologia, Avaliação, Observações, Parceiros, Equipe de Execução e Arquivos Anexos.

Figura 5.13: Página do SIPES ilustrando Sessões do módulo Projetos de Pesquisa.

Cada passo possui grupos (ex. “Dados Principais”) e cada grupo contém campos (ex. Título do Projeto, Edital e Área de Conhecimento). No exemplo da Figura 5.12, a proposta deste módulo está no estado inicial do *workflow*. Neste estado o grupo de usuários pesquisador pode criar propostas e editá-las. Os demais grupos não pertencem a nenhuma visão deste estado e, portanto, enquanto a proposta estiver neste estado somente usuários do grupo Pesquisadores poderão interagir com ela. Ao clicar em “Enviar proposta para julgamento” ela mudará de estado avançando no *workflow*.

Depois de listados todos os estados com seus respectivos módulos, o usuário deve escolher uma visão para ser executada. Ao escolher a visão, será exibido o estado do formulário que pertence ao usuário. Por exemplo: este formulário poderia ser um formulário de projeto de Iniciação Científica. Logo, ao

escolher a visão devem ser exibidos todos os projetos do usuário naquela visão, podendo disponibilizar as operações de editar, criar, apagar e visualizar, dependendo das permissões concedidas ao grupo de usuários ao qual o usuário pertence. Este processo é baseado na ação de listar registros de um formulário, esta ação pertence a seção que representa um módulo.

Um formulário no contexto da Aplicação Fênix é dividido em passos, cada passo de um formulário contém um conjunto de grupos e cada grupo é um conjunto de campos e, finalmente, cada campo é um atributo. Logo, cada passo de um formulário será correspondente a uma ação de visualizar, editar ou apagar, dependendo da escolha de operação feita pelo usuário.

A persistência dos dados é mantida por meio de um banco de dados que é criado juntamente com a aplicação para o SGBD PostgreSQL, conforme processo descrito na Seção 4.3.3 (O Módulo *Script Ger*) do Capítulo 4. Na Figura 5.14 pode-se visualizar uma representação lógica da arquitetura criada para o SIPES.

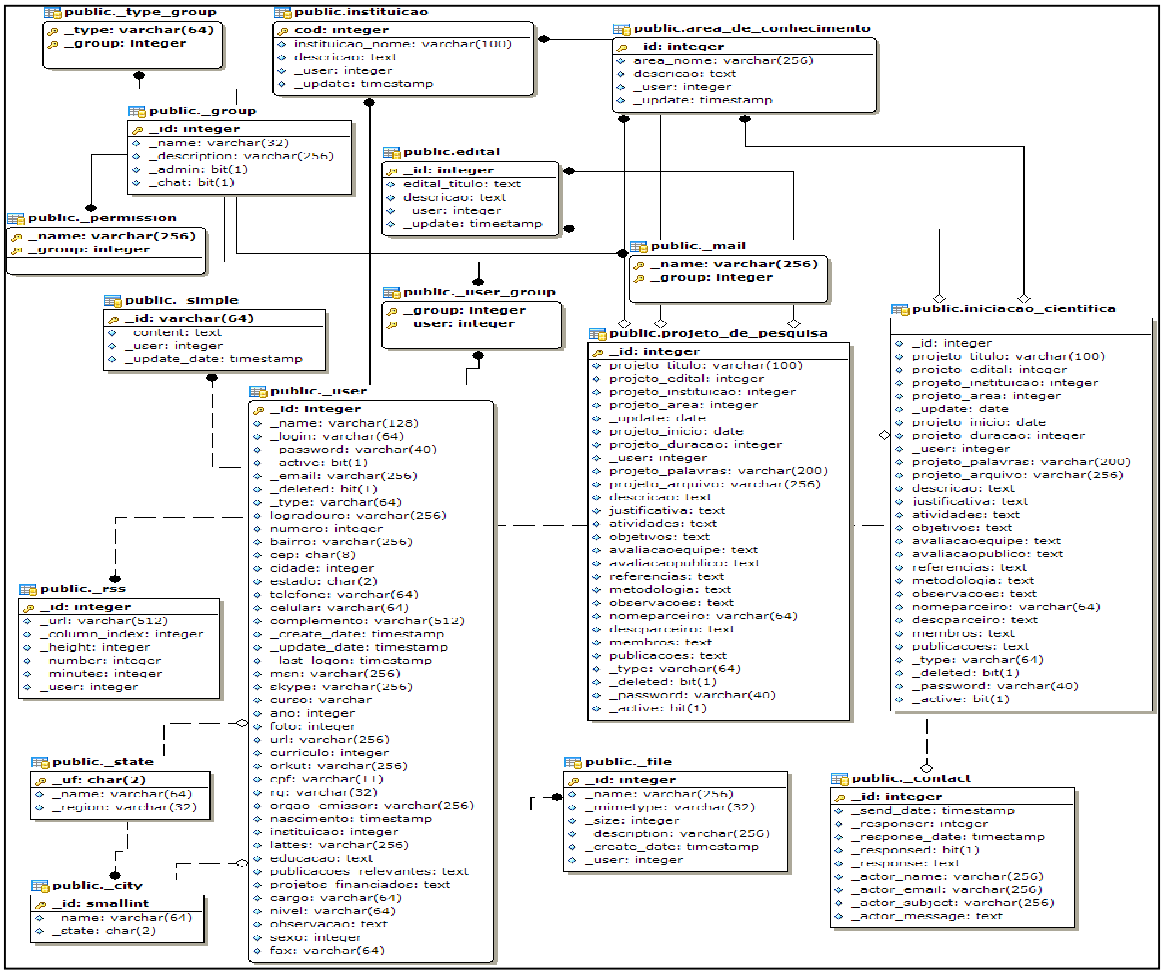


Figura 5.14: Modelo lógico do Banco de Dados Criado para o SIPES.

5.4. Arquitetura da *WebApp* Gerada

A aplicação gerada é um sistema baseado na *Web* e desenvolvido usando a arquitetura de três camadas: Apresentação, Aplicação e Persistência. A cada solicitação do usuário as páginas são geradas pela Camada de Aplicação e lhes são apresentadas por meio de um navegador *Web*. O armazenamento e a recuperação das informações no/do sistema é realizada com a solicitação à Camada de Apresentação. Essa por sua vez, comunica-se com a Camada de Aplicação que se comunica com a Camada de Persistência. A representação da arquitetura da aplicação é mostrada na Figura 5.15.

A Camada de Apresentação contém todos os elementos de interface e é a única camada visível aos usuários. Essa camada exibe o estado atual do sistema ao usuário, permite a entrada e saída de dados, a navegação, a propagação de eventos gerados pelo usuário e a requisição de páginas ou a propagação de tarefas que devem ser processadas na camada de aplicação.

Já a Camada Aplicação é responsável pela lógica do negócio, o que inclui algoritmos e regras procedimentais. Ela define o comportamento do sistema, ou seja, nessa camada intermediária as requisições são interpretadas e processadas e, em seguida, as respostas são enviadas para a maquina cliente.

Finalmente, a Camada de Persistência é responsável pelo acesso, criação ou destruição de algum objeto armazenado na base de dados, utilizando comandos SQL. Essa camada corresponde à lógica de manipulação de dados e de conexão com o SGBD e é responsável pelo armazenamento físico dos objetos do domínio em uma base permanente.

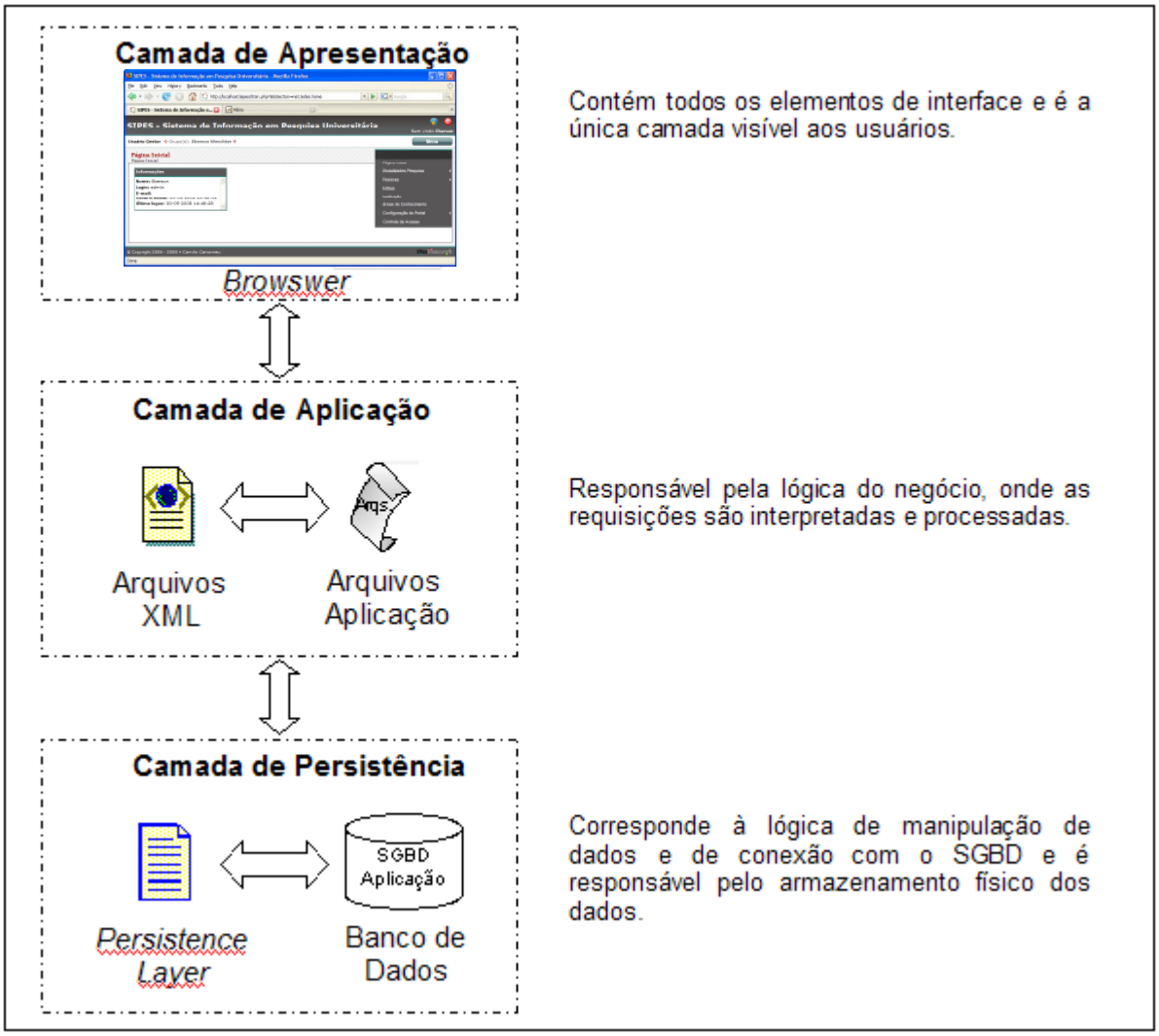


Figura 5.15: Arquitetura da aplicação gerada.

5.5. Considerações Finais

Neste capítulo foi apresentado um estudo de caso do processo de geração de uma aplicação *Web*, no domínio proposto, a partir da ferramenta Fênix. O Sistema de Informação em Pesquisa Universitária (SIPES) foi implementado segundo os padrões definidos no Fênix. Foi descrita a arquitetura geral da *WebApp* instanciada, a qual se constitui em um modelo em três camadas: Apresentação, Aplicação e Persistência. O próximo capítulo é destinado às considerações finais acerca do trabalho, abordando as conclusões do autor, bem como suas contribuições e sugestões de continuidade em trabalhos futuros.

Capítulo 6
Conclusões

6.1. Considerações Iniciais

O aumento considerável de aplicações baseadas na Web (*WebApps*) utilizadas nas organizações constitui-se em fator relevante para motivar a pesquisa e desenvolvimento de ferramentas apoiadas em técnicas e mecanismos que reusem artefatos de software já produzidos, auxiliando na geração automática de código e, por conseqüência, proporcionando desenvolvimento ágil das aplicações.

Dessa forma, a automatização do processo de desenvolvimento de software pode ser facilitada com uso de geradores de aplicações, que são ferramentas capazes de obter especificações em alto nível e gerar produtos de software e artefatos (documentação, *scripts* de criação de tabelas em banco de dados, código fonte, diagramas, entre outros).

Dentro deste contexto, no presente trabalho foi especificada e implementada a arquitetura para um gerador automático de *WebApps* baseado no *framework* Titan e integrado à ferramenta Fênix, que auxilia no processo de desenvolvimento de aplicações no domínio de Sistemas Web de Apoio à Gestão de Fomento de Projetos.

Para atingir os objetivos deste trabalho foi realizada uma revisão bibliográfica sobre modelos, ferramentas e as principais contribuições teóricas existentes sobre geradores de aplicações e geradores de código. A partir destes métodos foi possível analisar a ferramenta Fênix e propor uma arquitetura para seu módulo gerador integrado ao *framework* Titan. Neste capítulo são apresentadas as conclusões gerais deste trabalho, mostrando as principais contribuições (Seção 6.2), sugestões para trabalhos futuros (Seção 6.3) e dificuldades encontradas (Seção 6.4).

6.2. Contribuições

A principal contribuição deste trabalho foi mostrar a viabilidade da integração de geradores de aplicação e framework no contexto de reuso de Linha de Produtos de Software. A especificação e implementação do gerador de aplicação, gerador de código e a integração com o *framework* Titan no contexto da ferramenta Fênix é a principal contribuição deste trabalho. Dando continuidade a outro trabalho de mestrado do DCT/UFMS, foi proposta uma arquitetura para o gerador de aplicação para instanciação de *WebApps* no domínio SAGF, constituindo uma alternativa para atender a demanda no processo de desenvolvimento de aplicações pertencentes a esse domínio.

Destacam-se, também, a especificação e implementação do *wizard* para facilitar a geração de código e definição dos passos necessários para modelagem e geração de aplicação no domínio SAGF, utilizando, como infra-estrutura tecnológica básica a ferramenta Fênix.

O gerador de aplicações tem uma arquitetura baseada em três camadas e foi especificada para facilitar futuras extensões e adaptações. Foi implementado em Java e utilizada também, tecnologia XML, garantindo assim uma maior portabilidade da aplicação. Utiliza o banco de dados relacional PostgreSQL para armazenar a estrutura do projeto da *WebApp* gerada que pode ser visualizada em navegador Web do cliente.

Para validar e testar o gerador de aplicação foi gerado código em PHP e banco de dados PostgreSQL para uma aplicação simples de avaliação de projetos de pesquisa e iniciação científica (SIPES) da Pró-Reitoria de Pesquisa e Pós-Graduação da UFMS.

Como resultado, pode-se concluir que a ferramenta simplificou, organizou e orientou o projetista no desenvolvimento da aplicação. A facilidade na modelagem e na geração de aplicações de uma família de produtos que utilizam uma grande quantidade de dados foi uma característica relevante. A ferramenta de povoamento do gerador de aplicação facilitou o trabalho do administrador na manutenção da *WebApp* (povoamento de tabelas tais como cidades e estados). As aplicações geradas têm tamanho (Kbytes) pequeno e o carregamento das páginas é bastante rápido.

Além dos trabalhos e contribuições já apresentados, este trabalho contribuiu para o fortalecimento da linha de pesquisa em reutilização de software na área de *WebApps* no LEDES-DCT/UFMS, resultando em várias publicações em eventos nacionais e internacionais (WESCHTER, TURINE, 2007; WESCHTER et al., 2007; WESCHTER, TURINE, 2008a; WESCHTER, TURINE, 2008b).

6.3. Trabalhos Futuros

A partir dos resultados deste trabalho, vislumbram-se diversas pesquisas que podem ser abordadas em trabalhos futuros:

- O suporte de novos tipos de atributos na ferramenta Fênix exige uma re-implementação. Essa limitação deu origem à idéia de criar, como trabalho futuro, uma nova arquitetura do gestor de atributos baseada em engenharia de componentes. Indubitavelmente, isso aumentaria a flexibilidade quanto a necessidade de diferentes tipos de campos de formulários que uma aplicação instanciada poderia utilizar.
- Extensão da arquitetura do Fênix para possibilitar a interoperabilidade com outros sistemas, por exemplo, currículo Lattes do CNPq a fim de fornecer informações para auxiliar na tomada de decisão dos gestores.
- Desenvolvimento de um módulo de conexão e integração a banco de dados independente no *Framework* Titan, possibilitando, dessa forma que o Engenheiro de Aplicação tenha opção de escolher qual sistema de banco de dados irá utilizar para a aplicação instanciada.
- Desenvolvimento de *WebApps* personalizadas, ou seja, aplicações que atendem as necessidades individuais de cada usuário, é um grande desafio. Esta personalização é um aspecto importante em vários tipos de aplicação. A investigação de como desenvolver tais aplicações utilizando o ambiente Fênix pode ser escopo de um trabalho futuro.
- Gestão dinâmica de documentos de estilo de interface XSLT a fim de facilitar visualização dos diferentes produtos de software.
- Reimplementar o Gestor de Atributos da ferramenta Fênix de forma a criar Tipos como componentes EJB (*Entreprise Java Beans*). Atualmente o

suporte de novos tipos de atributos na ferramenta Fênix exige re-implementação. Essa limitação deu origem à idéia de criar, como trabalho futuro, uma nova arquitetura do gestor de atributos baseada em engenharia de componentes. Indubitavelmente, isso aumenta a flexibilidade quanto as necessidades de diferentes tipos de campos de formulários que uma aplicação instanciada poderia utilizar;

- Outro possível trabalho seria a integração automática de testes das aplicações geradas por meio da ferramenta Fênix, integrada ao framework Titan. As atividades de geração de casos de testes poderiam ser integradas, de forma que ao gerar uma aplicação, fossem gerados automaticamente os casos de testes associados, seguindo algum critério de testes previamente especificado;
- Realização de mais experimentos e casos de uso envolvendo a geração automática de *WebApps* em diferentes modelos dentro do domínio de aplicação proposto são necessários para a sua validação.

6.4. Dificuldades

Um dos problemas que o autor deste projeto enfrentou inicialmente no mestrado foi a adaptação em conciliar os estudos e pesquisas com extensa carga de trabalho profissional. Porém, as principais dificuldades estão relacionadas à compreensão das diferentes abordagens existentes de geradores de código e aplicações, e assimilação das mesmas no intuito de criar uma arquitetura para o módulo gerador da ferramenta Fênix.

REFERÊNCIAS BIBLIOGRÁFICAS

AHO, Alfred V., RAVI, S., ULLMAN, Jeffrey, D. Compiladores: princípios, técnicas e ferramentas. Rio de Janeiro: LTC, 1995.

ANDROMDA. AndroMDA Tool. Disponivel em: http://www.andromda.org. [20/04/2006], 2006.

ARAGON, C. R. Processo de Desenvolvimento de uma Linha de Produtos para Sistemas de Gestão de Bibliotecas. (Dissertação de Mestrado). Departamento de Computação e Estatística, UFMS. Campo Grande, Mato Grosso do Sul, Brasil. 2004

BALASUBRAMANIAN, V.; BASHIAN, A.; PORCHER, D. A large-scale hypermedia application using document management and Web technologies. *In: Proceedings of Hypertext'97, VIII International ACM Hypertext Conference, Southampton*, UK, 1997.

BALZERANI, L.; DI RUSCIO, D.; PIERANTONIO, A.; DE ANGELIS, G. A. Product line architecture for web applications. *ACM Symposium on Applied Computing*. Santa Fé, Nex México, 2005.

BRAMBILLA, M.; CERI, S.; FRATERNALLI, P.; MANOLESCU, I. Process modeling in Web applications. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol 15, No 4, Oct. 2006.

BARRÉRE T. S. CASE com Múltiplas Visões de Requisitos de Software e Implementação Automática em Java - MVCASE. Dissertação de Mestrado. Universidade Federal de São Carlos. São Carlos, São Paulo, Brasil, 1999.

BARRY B. Managing Software Productivity and Reuse. IEEE Computer, vol. 32, nº. 9, Sept., 1999.

BASS, P.; CLEMENTS, L.; KAZMAN, R. Software Architecture in Practice. Boston: Addison-Wesley, Second Edition, 2002.

BATORY, D. S.; LOPEZ-HERREJON, R.; MARTIN J. P. Generating product-lines of product-families. In *Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE International Conference*, 2002.

BATORY, D. S. Feature models, grammars, and propositional formulas. In *Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005, Proceedings*, volume 3714 of *Lecture Notes in Computer Science*, Springer, 2005.

BAUER, C.; KING, G. Hibernate in Action. Manning Publications, 2005.

BENGTSSON, P. O.; BOSCH, J.; MOLIN, P.; MATTSSON, M. Object oriented *framework* - problems & experiences. In: FAYAD, M.; JOHNSON, R.; SCHMIDT, D. (Ed.). Building

Application *Framework*s: Object-Oriented Foundations of *Framework* Design. Wiley & Sons, 1999.

BHATTI, R.; GHAFOOR, A.; BERTINO, E.; JOSHI, J. B. D. X-GTRBAC: An XML-Based Policy Specification Framework and Architecture for Enterprise-Wide Access Control. *ACM Transactions on Information and System Security, Vol. 8, No. 2*, May 2005.

BIEBER, M.; GALNARES, R.; LU, Q. Web Engineering and flexible hypermedia. *Proceedings of the 2nd Workshop in Adaptive Hypertext and Hypermedia, Hypertext*, 1998.

BOSCH, J. Software product lines: organizational alternatives. *Proceedings of the 23rd international conference on Software Engineering*. Toronto, Ontario, Canada: IEEE Computer Society, 2001.

BOSCH, J.; BENGTSSON, P. O.; MOLIN, P.; MATTSSON, M. Object oriented *framework* - problems & experiences. Wiley & Sons, 1999.

BRAGA, R. T. V. Um Processo para Construção e Instanciação de *Framework*s baseados em uma Linguagem de Padrões para um Domínio Específico. (Tese de Doutorado). Instituto de Ciências Matemáticas e de Computação da USP-SC, São Carlos, São Paulo, Brasil, 2003.

BUSCHMANN, F., MEUNIER, R., ROHNERT, H. Pattern-Oriented Software Architecture, A System of Patterns, 1 ed., John Wiley & Sons, 1996.

CARROMEU, C.; TURINE, M. A. S. Um *Framework* de Aplicações Web de Apoio a Gestão de Fomento. Workshop de Teses e Dissertações do XI Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia 2005). Poços de Caldas, Minas Gerais, Brasil, 2005.

CARROMEU, C. Linha de Produtos de Software no Processo de Geração de Sistemas Web de Apoio a Gestão de Fomento de Projeto. (Dissertação de Mestrado). Departamento de Computação e Estatística, UFMS. Campo Grande, Mato Grosso do Sul, Brasil, 2007.

CARROMEU, C.; TURINE, M. A. S. (2006a). Processo de uma Linha de Produtos para Sistemas Web de Apoio a Gestão de Fomentos. XXXII Conferencia Latinoamericana de Informática (CLEI 2006). Santiago do Chile, 2006.

CARROMEU, C.; TURINE, M. A. S. (2006b). Fênix: Uma Ferramenta para Automatizar o Processo de Linha de Produtos de Software no Domínio de Gestão de Fomento. XXXIII Seminário Integrado de Software e Hardware (SEMISH 2006). Evento integrante do XXVI Congresso Brasileiro da Sociedade Brasileira de Computação. Campo Grande, MS, Brasil, 2006.

CLEMENTS, P.; NORTHROP, L. Software Product Lines: Practices and Patterns. Boston: Addison-Wesley, 2002.

CLEAVELAND, J. C. Building Application Generators. *IEEE Software 5*, jul, 1988.

CLEAVELAND, J. C. Program Generator with XML and Java. Prentice Hall, 2001.

COALLIER, F. A Vision for International Standardization in Software and Systems Engineering. *6th International Conference on the Quality of Information and Communications Technology* (QUATIC 2007). Universidade Nova de Lisboa. Lisboa, Portugal, 2007.

COHEN, S. G.; HESS, J. A.; KANG, K. C.; NOVAK, W. E.; PETERON, A. S. Feature Oriented Domain Analysis (FODA) Feasibility Study. Pittsburgh, Pennsylvania, USA, 1990.

COHEN, S. Product Line State of the Practice Report, 2002. Disponível em: http://www.sei.cmu.edu/publications/documents/02.reports/02tn017.html. [31/08/2006].

CONALLEN, J. Building Web applications with UML. Addison Wesley Object Technology Series, 2000.

CONVERSE, T.; PARK, J. PHP: a Bíblia. Editora Câmpus, 2003.

CRNKOVIC, I.; HNICH, B.; JONSSON, T.; KIZILTAN, Z. Specification, implementation, and deployment of components. *Communications of the ACM*, ACM Press, 2002.

CZARNECKI, K. Overview of Generative Software Development. In Unconventional Programming Paradigms (UPP), Mont Saint-Michel, France, LNCS 3566, 2004.

CZARNECKI, K.; EISENERCKER, U. W. Generative Programming. Addison-Wesley, 2002.

DELTA SOFTWARE. Generative Programming: From Theory to Practice. Disponível em: http://www.d-s-tg.com/neu/media/pdf/Facts_e/gp_hintergrnd.pdf, [17/05/2007].

DENGER, C.; KOLB, R. Testing And Inspecting Reusable Product Line Components: First Empirical Results**.** *ISESE'06: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering.* Rio de Janeiro, Brazil, 2006.

DISTANTE, D.; ROSSI, G.; CANFORA, G. Modeling business processes in web applications: an analysis framework. *Proceedings of the 2007 ACM symposium on Applied computing*, Seul, Korea, 2007.

ECKEL, B. Thinking in Java. Prentice Hall, 2003.

EISENBERG, J. D.. SVG Essentials. O'Reilly, 2002.

FAYAD, M. Object-oriented application frameworks. *Communication of the ACM*, 1997.

FERTALJ, K.; KALPIC, D.; MORNAR, V. Source code generator based on a proprietary specification language. *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS-35.02). IEEE Computer Society*, 2002.

FOOTE, B.; JOHNSON, R. E. Designing reusable classes. *Journal of Object Oriented Programming,* 1988.

FRAKES, W., KANG, K. Software Reuse Research: Status and Future. *IEEE Transactions on Software Engineering*, 2005.

FRANCA, L. P. A. Um Processo para a Construção de Geradores de Artefatos. (tese de doutorado). Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, RJ, Brasil, 2000.

FRANCA, L. P. A.; STAA, A. V. Geradores de Artefatos: Implementação e Instanciação de *Framework*s. In: Anais do XV SBES-2001- Simpósio Brasileiro de Engenharia de Software. Rio de Janeiro, Brasil, 2001.

GAMMA, E., HELM, R., JOHNSON, R. Padrões de Projeto – Soluções Reutilizáveis de Software Orientado a Objetos. Ed. Bookman, 1995.

GILL, N. S. Importance Of Software Component Characterization For Better Software Reusability. *ACM SIGSOFT Software Engineering Notes, Vol 31, No 1*, july 2006.

GIMENES, I.M.D.S., HUZITA, E.H.M. Desenvolvimento Baseado em Componentes. Rio de Janeiro, Ciência Moderna, 2005.

GIMENES, I. M. S.; TRAVASSOS, G. H. O enfoque de linha de produto para desenvolvimento de software. In: Evento integrante do XXII Congresso da SBC - SBC2002. XXI Jornada de Atualização em Informática. Sociedade Brasilera de Computação. Florianópolis, Santa Catarina, 2002.

GLOVER, E.; TSIOUTSIOULIKLIS, K.; LAWRENCE, S.; PENNOCK, D.; FLAKE G. Using Web structure for classifying and describing Web pages. *Proceedings of WWW2002, International Conference on the World Wide Web*, 2002.

GOAER, O.; TAMZALIT, D.; OUSSALAH, M. C.; SERIAI, A. D. Evolution styles to the rescue of architectural evolution knowledge. *Proceedings of the 3rd international workshop on Sharing and reusing architectural knowledge*, Leipzig, germany, 2008.

GREVE, F.G.P., ARAÚJO, J.G.R., ANDRADE, S.S., MAIA FILHO, E.M.F., BRITO, K.S., ROCHA, L.A., PINHEIRO, V.G. Cordel: uma Ferramenta Distribuída para a Geração de Aplicações Web*. In I2TS 3rd International Information and Telecommunication Technologies Symposium*. São Carlos, São Paulo, Brasil, 2004.

HERRINGTON, J. Code Generation in Action. Manning, Greenwich, CT, 2003.

HUSTED, T., DUMOULIN, C., FRANCISCUS, G., WINTERFELDT, D. Struts in Action - Building web applications with the leading Java *framework*. Manning Publications, 2003.

ISHY, E. Uma ferramenta Web para gestão de FAQ utilizando a abordagem de componentes ADComp. (Dissertação de Mestrado). Departamento de Computação e Estatística, UFMS. Campo Grande, Mato Grosso do Sul, Brasil, 2004.

JARZABECK, S. From reuse library experiences to application generation architectures*. Symposium on Software Reusability (SSR' 95)*. ACM-SIGSOFT, 1995.

JOHNSON, R. E. Documenting *framework*s using patterns. *In: Conference proceedings on Object-oriented programming systems, languages, and applications*. Vancouver, British Columbia, Canada: *ACM Press*, 1992.

KRUEGER, C. W. Software Reuse. *ACM Computing Surveys 24*, 1992.

LAI, C. T. R.; WEISS, D. M. (1999). Software Product-Line Engineering: A Family- Based Software Development Process. Addison-Wesley, 1999.

LAU, K. Software Component Model. *Proceedings of the 28th international conference on Software engineering*, Shanghai, China, 2006.

LEDES. Titan - um *framework* de aplicações Web. Disponível em: http://www.ledes.net/titan/ [18/06/2006], 2006.

LIM, W. C. Managing Software Reuse*:* A Comprehensive Guide to Strategically Reengineering the Organization for Reusable Components*. Prentice Hall*, Inc. Upper Saddle River, NJ, 1998.

LUCRÉDIO, D.; ALVARO, A.; ALMEIDA, E. S. de; PRADO, A. F. Do. MVCASE Tool - Working with Design Patterns. In: The Third Latin American Conference on Pattern Languages of Programming. *Proceedings of The Third Latin American Conference On Pattern Languages Of Programming*. Porto de Galinhas, BA, 2003.

LUKER, P. A. Program Generators and Generation Software*. The Computer Journal,* No. 29 (4), 1986.

MOREIRA, D., MRACK, M. Sistemas Baseados e Metamodelos. In: II WCOMPI – Workshop de Computação e Gestão da Informação. UNIVATES, Lageado, RS, Brasil, 2003.

OLIVEIRA, M.; BACILLI, K. O Reuso na Prática: O reuso como diferencial competitivo em produtividade e qualidade no desenvolvimento de software. Mundo Java No 18, julho/agosto, 2006.

PAIS, A.P.V.; OLIVEIRA, C.E.T.; LEITE, P.H.P.M. Robustness Diagram: A Bridge Between Business Modeling And System Design *. In: Proceedings of VII International Conference on Object-Oriented Information Systems - OOIS'01*. Calgary, Canadá: Springer- Verlag, 2001.

PAZIN, A. GawCRe: Um Gerador de Aplicações baseadas na Web para o Domínio de Clínicas de Reabilitação. (Dissertação de Mestrado). Centro de Ciências Exatas e de Tecnologia, UFSCar, São Carlos, São Paulo, Brasil, 2004.

PRADO, A. F., LUCRÉDIO, D. Ferramenta MVCASE - Estágio Atual: Especificação, Projeto e Construção de Componentes. Sessão de Ferramentas do XV Simpósio Brasileiro de Engenharia de Software - SBES'2001. Rio de Janeiro-RJ, Brasil. Outubro, 2001.

PREE, W. Design Patterns for Object-Oriented Software Development. Addison-Wesley, 1995.

PREE, W.; SIKORA, H. Design patterns for object-oriented software development (tutorial). *In: Proceedings of the 19th international conference on Software engineering*. Boston, Massachusetts, USA: *ACM Press*, 1997.

ROCHA, L. S.; NOGUEIRA, R.; PRUDÊNCIO, J. G.; ANDRADE, R. M. C; SOUZA, J. T. XSpeed: Uma ferramenta para geração de aplicações distribuídas baseadas em padrões, 2004. Disponível em: http://www.lia.ufc.br/~great/artigos.htm. [30/08/2006].

ROSS, D. *Structured analysis: A language for communicating ideas*. IEEE Transmition Software Engineering, v. 3, 1977.

ROSSI, G.; SCHWABE, D.; GUIMARÃES, R.M. *Designing personalized web applications*. International World Wide Web Conference (WWW), Amsterdam, 2001.

SAMETINGER, J. Software Engineering with Reusable Components. Springer-Verlag New York, Inc, 1997.

SANTOS, E. H. CodeCharge: Gerador de Códigos para Aplicações Web. EMBRAPA. Comunicado Técnico numero 45. Campinas, São Paulo, Brasil, 2002.

SHIMABUKURO, E. K. Um Gerador de Aplicações Configurável. (Dissertação de Mestrado). Instituto de Ciências Matemáticas e de Computação da USP-SC, São Carlos, São Paulo, Brasil, 2006.

SMARAGDAKIS, Y.; BATORY, D. Application Generators. Department of Computer Sciences. The University of Texas at Austin, 1998. Disponível em :http://www.cc.gatech.edu/~yannis. [31/07/2006].

SMARAGDAKIS, Y.; BATORY, D. Application Generators. *Encyclopedia of Electrical and Electronics Engineering,* j.G. Webster (ed.), Jhon Wiley and Sons, 2000.

SOMMERVILLE, I. Software Engineering, 8th Edition. Addison-Wesley, 2006.

STEARNS, B.;MURRAY, G.; SINGH, I.; INSCORE, J.; DEMICHIEL, L.; JOHNSON, M.; KASSEM, N.; SHARMA, R.; ORTIGAS, R.; MONZILLO, R.; BRYDON, S.; RAMACHANDRAN, V. Designing Enterprise Applications with the J2EE Platform. 2th, Addison-Wesley, 2002.

STINSON, B. PostgreSQL Essential Reference. Sams Publishing, 2001.

SUN. Enterprise javabeans technology, 2006. Disponível em:http://java.sun.com/products/ejb/ [15/06/2006].

TRACZ, W. *DSSA* (Domain-Specific Software Architecture) Pedagogical Example*. ACM SIGSOFT Software Engineering Notes*, 2005.

TURINE, M.A.S.; ISHY, E. Gestão de FAQ em Ambientes de EAD utilizando a Ferramenta Baseada em Componentes INSTANTFAQ. In: Conferência IADIS Ibero-WWW/Internet 2004. Madrid: IADIS Press, 2004.

TURINE, M.A.S.; VIEIRA, C.C.; ARAKAKI, A.A.; SANDIM, H.C. (2005a). Pantaneiro: Um Gerador de Aplicações Web Baseadas em Componentes. In: XI Simpósio Brasileiro de Sistemas Multimídia e Web. Poços de Caldas, Minas Gerais, Brasil, 2005.

TURINE, M.A.S.; LIMA, J.E.O; ROMÃO, R. (2005b). Uma Arquitetura Baseada em Componentes para um Sistema de Informação em Gestão Social. In: XI Simpósio

Brasileiro de Sistemas Multimídia e Web - Webmedia 2005. Poços de Caldas, Minas Gerais, Brasil, 2005.

VITHARANA, P., ZAHEDI, F., JAIN, H. (2003). Design, Retrieval, and Assembly in Component-based Software Development. *Communication of the ACM*, 2003.

W3C. Sha-1 - secure hash algorithm, 1998. Disponível em: http://www.w3.org/TR/1998/REC-DSig-label/SHA1-1 0 [25/03/2006].

WANG, Z.; HABERL, W.; KUGELE, S.; TAUTSCHNIG, M. Automatic generation of systemc models from component-based designs for early design validation and performance analysis. *WOSP '08: Proceedings of the 7th international workshop on Software and performance*. Princeton, NJ, USA, 2008.

WEBER, K. C.; NASCIMENTO, C. J.; Brazilian Software Quality 2002. *The 24th IEEE International Conference on Software Engineering (ICSE)*, 2002.

WESCHTER, E. O.; TURINE, M. A. S. Uma Proposta de Arquitetura para o Gerador de Aplicação Web no Domínio de Gestão de Fomento de Projetos. I Conferência Internacional de Educação Profissional e Tecnológica, Cuiabá/MT, Brasil, 2007.

WESCHTER, E. O.; TURINE, M. A S.; CARROMEU, C. Arquitetura de um Gerador De Aplicações Web Para Agências De Fomento No Brasil. *International Conference On Engineering And Computer Education*, Santos-SP, 2007.

WESCHTER, E. O.; TURINE, M. A. S. FÊNIX: (2008a). A Model Architecture of a Web Application Generator For Fomentation Agencies in Brazil*. In: X International Conference On Engineering and Technology Education - INTERTECH'2008*, Peruíbe, Brazil. 2008.

WESCHTER, E. O.; TURINE, M. A. S. (2008b). Architecture of a Web Application Generator For Agencies of Fomentation In Brazil. *In: International Technology, Education and Devlopment Conference - INTED 2008*, Valencia, Spain. 2008. (Acepted).