# Instanciação de Sistemas de Informação para Educação a Distância com o Framework Titan

**MSc. Camilo Carromeu**

Campus de Coxim – Universidade Federal de Mato Grosso do Sul (UFMS)

Caixa Postal 549 – 79070-900 – Campo Grande – MS – Brasil

camilo@carromeu.com

***Resumo*** *Este artigo propõe a criação de sistemas de informação para educação a distância utilizando técnicas de reusabilidade de software, tais como frameworks e padrões. A proposta apresenta um ambiente de reuso de artefatos de software de forma que tais sistemas possam ser criados de forma rápida e com maior qualidade. Para criação deste ambiente é utilizado um framework de instanciação de sistemas gerenciadores de conteúdo (Content Management System - CMS), desenvolvido pelo LEDES/DCT/UFMS, denominado Titan.*

## 1. Introdução

Os programas de educação a distância usam, necessariamente, tecnologias para suplantar ou substituir as instruções ao vivo, face-a-face. As tecnologias devem preferencialmente ser usadas para proporcionar aos estudantes a oportunidade de interagir e trabalhar juntos em problemas e projetos significativos, e juntar-se a comunidades de alunos e profissionais [Selfe 1988] [Bales 1990] [Seaton 1993] [Nalley 1995]. A tecnologia deve estender o melhor das práticas em sala de aula para localidades distantes, ao invés de reproduzir o pior [Burge 1993].

Diversas tecnologias digitais, analógicas e materiais impressos podem ser utilizadas para este fim, dentre as quais destacam-se a TV, o vídeo e a Internet. Estas tecnologias se destacam, principalmente, pela sua natureza multimídia, ou seja, integram a mídia tal como

texto, som, gráficos, animação, vídeo, imagem e a modelagem espacial. Em sistemas computacionais, com o advento dos monitores de alta resolução, som e cartões de compressão de vídeo, memória de acesso casual para computadores pessoais, a multimídia tem sido condensada somente em um box. O computador de mesa é capaz, agora, de captar sons e vídeo, gerar todos os tipos de gráficos, incluindo animação, e integra-os todos dentro de uma única apresentação multimídia. Indivíduos com pouca experiência podem tornar-se seus próprios artistas, editores ou produtores de vídeo.

A hipermídia é derivada do hipertexto, que é um método não-seqüencial, não-linear para organização e exibição de texto [Jonassen 1989]. Foi projetada para permitir ao leitor o acesso à informação de um texto em formas que sejam mais significativas para ele. A hipermídia consiste em pontos centrais (trechos ou fragmentos de texto, figuras, animações, sons ou documentos). Ao estudar uma base de conhecimento hipermídia, os usuários podem acessar pontos centrais em qualquer ordem que os satisfaça. Em muitos sistemas de hipermídia, os pontos centrais podem ser emendados ou modificados pelo usuário, de modo que o sistema possa ser uma base dinâmica de conhecimento que continue a crescer, representando novos e diferentes pontos de vista.

Sistemas de informação baseados em gestão de conteúdo proporcionam uma forma unificada de estabeler comunidades de alunos e profissionais e, ao mesmo tempo, possibilitam a disponibilização de conteúdo multimídia e hipertexto a esta comunidade. A criação de sistemas de informação no domínio de educação a distância não é uma tarefa trivial, já que deve levar em consideração subsistemas eficazes de log, controle de permissões, ferramentas de comunicação inter-usuários, métricas de produção de alunos, dentre muitos outros. Tal problemática pode ser confrontada através de técnicas de reuso de software.

Muitos produtos de software são desenvolvidos em função de artefatos já especificados e implementados, utilizando técnicas de reutilização de software para melhorar a produtividade, a manutenibilidade e a qualidade tanto do software quanto do processo do desenvolvimento. O reuso caracteriza-se pela utilização de artefatos de software numa situação diferente daquela para a qual foram originalmente construídos [Pressman 2005] [Cheesman e Daniels 2001]. As técnicas de reutilização usualmente não

representam um sistema completo e sim especificações e/ou implementações de soluções parciais, tornando o processo de reuso uma atividade apenas de parametrização de variáveis.

É consenso na literatura 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 [Bengtsson et al 1999] [Depaez 1999] [Gimenes e Travassos 2002] [Braga 2003] [Balzerani et al 2005]. Diversas técnicas de reuso têm sido propostas na literatura, tais como, engenharia de domínio [Sommerville 2001], *frameworks* [Johnson 1992] [Fayad e Schmidt 1997] [Fayad, Johnson e Schmidt 1996] [Pree 1999] [Braga, Germano e Masiero 1999] [Brugali e Sycara 1999] [Braga 2003], padrões [Frye e Yoder 2001] [Buschmann et al 1996], linguagem de padrões [Gamma et al 1993] [Gamma et al 1995] [Fayad 1999] [Braga 2003] [Aragon 2004], desenvolvimento baseado em componentes (DBC) [Dsouza e Wills 1999] [Crnkovic et al 2002], geradores de aplicação [Franca e Staa 2001] [Batory e Smaragdakis 2003], linha de produtos de software (LPS) [Clements e Northrop 2001] [Lai e Weiss 1999] [Bosch 2001] e aspectos [Kiczales 1997] [Tarr 2000].

Tais características motivam a integração de técnicas e de ferramentas de reutilização de software no processo de desenvolvimento de Aplicações Web (WebApps) e, mais especificamente, na criação de sistemas de informação no domínio de educação a distância. Assim, este artigo apresenta uma proposta de utilização da técnica de *frameworks* para especificar uma maneira sistemática de reuso de software, permitindo criar um conjunto de produtos de software em um mesmo domínio de negócio com características similares, por meio da definição de uma infra-estrutura comum dos artefatos que compõem os produtos, e da parametrização das diferenças entre eles.

Este artigo apresenta uma proposta de um ambiente de desenvolvimento de novos membros para a família de produtos de software (FPS) do domínio de educação a distância baseado em reuso. Mais especificamente, o ambiente será baseado na geração de novas instâncias do *framework* Titan [Carromeu 2007], desenvolvido pelo Laboratório de Engenharia de Software (LEDES) do Departamento de Computação e Estatística (DCT) da Universidade Federal de Mato Grosso do Sul (UFMS), adaptado para este domínio. A partir

de uma arquitetura genérica comum e de um conjunto de artefatos que povoam a arquitetura, providos pelo Titan *Framework*, são desenvolvidos os produtos de software. Cada produto é, portanto, uma instância parametrizada do Titan.

A apresentação deste trabalho é organizada em 6 seções. A Seção 2 apresenta o reuso de software baseado em padrões e *frameworks*. Na seqüência, a Seção 3 descreve o Titan Framework e o componentes específicos que formam o ambiente de desenvolvimento de novas instâncias no domínio de educação a distância. Na seção 4, são apresentados os dois estudos de caso: o Sistema de Informação Acadêmico a Distância – SIAD e o Sistema de Cursos da Escola de Conselhos. A seção 5 apresenta a conclusão do trabalho e as perspectivas para trabalhos futuros. Finalmente, a seção 6 apresenta as referências bibliográficas.

## 2. Padrões e *Frameworks*

Reuso é uma estratégia de solução de problemas utilizada na maioria das atividades do ser humano. A reutilização de software surgiu no final dos anos 1960 como uma alternativa para superar a crise do software, e tem como objetivo principal desenvolver software com qualidade e economicamente viável utilizando artefatos já especificados, implementados e testados. Segundo Frakes e Terry [Frakes e Terry 1996], a reutilização de software pressupõe o uso de artefatos existentes ou o conhecimento para criação de novos softwares.

Com o objetivo de reuso em diferentes níveis de abstracão, surgiram na década de 90, os padrões de software [Buschmann *et al* 1996] [Coplien 1998] [Gamma *et al* 1995], que tentam captar a experiência adquirida no desenvolvimento de software e sintetizá-la em forma de problema e solução. Além de prover o reuso das soluções, os padrões ajudam a melhorar a comunicação entre desenvolvedores, que podem conduzir suas discussões com base na identificação dos padrões [Gamma *et al* 1995]. Para desenvolvedores que conhecem os padrões, a simples citação de seu nome traz consigo um conteúdo semântico significativo, o que dispensa a explicação dos detalhes envolvidos na solução.

Durante o processo de resolução de um problema particular, raramente os especialistas inventam uma nova solução diferente das já existentes. Diversas soluções são conhecidas de acordo com experiência própria ou de outros profissionais. Assim, ao confrontar-se com novos problemas, é comum recuperar e lembrar soluções antigas, pensando em pares "problema/solução" [Buschmann *et al* 1996] [Gamma *et al* 1995]. Os padrões de software descrevem soluções de processo para problemas que ocorrem com freqüência no desenvolvimento de software, e tem por função recuperar e documentar a experiência adquirida pelos desenvolvedores durante a prática profissional. Segundo Fayad, Johnson e Schmidt [Fayad, Johnson e Schmidt 1999], os padrões descrevem soluções recorrentes aprovadas durante seu tempo de uso, permitindo que sejam evidenciadas características comuns que podem ser reutilizadas em novos projetos e em diferentes níveis de abstração, não somente de código, mas de processo, de arquitetura, de análise, de projeto, de programação, dentre outros.

Os *frameworks* são definidos como aplicações semi-completas e reutilizáveis que, quando especializadas, produzem aplicações personalizadas dentro de um domínio específico [Foote e Johnson 1988]. Apesar de serem tipicamente voltados para o reuso, *frameworks* possuem características, como dependência de interfaces bem definidas, reuso de projeto e arquitetura, uso de padrões, que podem auxiliar no desenvolvimento de sistemas com arquiteturas mais organizadas para permitir maior facilidade de adaptação e extensão. A estrutura de um *framework* pode ser definida como um conjunto de classes que contém o projeto abstrato de soluções para uma família de problemas, propiciando o reuso com granularidade maior do que classes [Foote e Johnson 1988]. É composto por uma coleção de classes abstratas e concretas e de interfaces entre elas, representando o projeto de um subsistema [Sommerville 2001].

Os *frameworks* são compostos de partes fixas e partes variáveis. As partes fixas são denominadas pontos-fixos (*frozen-spots*) e são estáveis e imutáveis durante o uso do *framework* em diversas aplicações. As partes variáveis são denominadas pontos variáveis (*hot-spots*) e devem ser adaptadas de forma particular em cada instanciação do *framework* [Shimabukuro 2006].

De acordo com Fayad e Schmidt [Fayad e Schmidt 1997] os *frameworks* podem ser classificados, segundo seu escopo, em três grupos:

- *Frameworks* de infra-estrutura do sistema: simplificam o desenvolvimento da infra-estrutura de sistemas portáveis e eficientes como, por exemplo, sistemas operacionais, sistemas de comunicação, interfaces com o usuário e ferramentas de processamento de linguagem;

- *Frameworks* de integração *middleware*: geralmente usados para integrar aplicações e componentes distribuídos. São projetados para melhorar a habilidade de desenvolvedores em modularizar, reutilizar e estender a infra-estrutura de software para funcionar em um ambiente distribuído; e

- *Frameworks* de aplicação empresarial: voltados para domínios de aplicação amplos e são base para atividades de negócios das empresas, tais como, sistemas de telecomunicações, aviação, manufatura e engenharia financeira.

Considerando a forma de reuso, os *frameworks* podem ser classificados como *framework* caixa-branca (*whitebox*), caixa-preta (*blackbox*) ou caixa-cinza (*graybox*). As funcionalidades de um *framework* caixa-branca são reusadas e estendidas por meio de herança. Dessa forma, o usuário precisa conhecer detalhes das estruturas internas do *framework* para utilizá-lo. O *framework* caixa-preta permite estender suas funcionalidades por composição ou definição de interfaces para os componentes. Assim, o usuário deve entender apenas a interface para usar este tipo de *framework*. Finalmente, o *framework* caixa-cinza é uma combinação do caixa-branca e do caixa-preta, ou seja, o reuso é feito por herança e pela definição de interfaces. Ele deve possuir flexibilidade e extensibilidade suficiente, e a habilidade de ocultar de seus usuários informações desnecessárias [Fayad, Johnson e Schmidt 1999].

Como inicialmente é difícil conhecer todo o domínio da aplicação e prever todos os casos de uso para desenvolver um *framework* caixa-preta, as primeiras versões de um *framework* são, geralmente, caixa-branca. Na medida em que vai sendo utilizado o

*framework* é melhorado sistematicamente até ser transformado em um *framework* caixa-cinza e, posteriormente, caixa-preta [Fayad, Johnson e Schmidt 1999] [Yassin e Fayad 1999].

Resumidamente, um *framework* é um grande modelo de um domínio de aplicações e, desta forma, pode ser desenvolvido a partir de um conjunto de aplicações do domínio que atuam como fontes de informação deste domínio [Silva 2000]. As propostas de metodologias de desenvolvimento de *frameworks* mais relevantes no contexto deste artigo são:

- Projetos dirigidos por exemplos [Johnson 1993]: a abstração do domínio, que é o próprio *framework*, é obtida por meio de casos e sistema reais, ou seja, as aplicações em uma família de produtos;

- Projeto dirigido por *Hot Spot* [Pree 1994]: os *hot spots* são as partes flexíveis do *framework*, e a essência desta metodologia é identificar os *hot spots* na estrutura de classes de um domínio e, a partir disto, construir o *framework*; e

- Projeto dirigido por Linguagem de Padrões [Braga 2003]: nesta abordagem o desenvolvimento de um *framework* tem como base uma linguagem de padrões em um particular domínio.

O desenvolvimento do *framework* utilizado no ambiente de desenvolvimento apresentado neste artigo foi baseado no modelo dirigido por exemplos, permitindo, durante a sua fase de evolução, abstrair os padrões a partir de sistemas no domínio da família de produtos de educação a distância.

## 3. Titan Framework Aplicado ao Domínio de Educação a Distância

O Titan é um *framework* de aplicações criado e desenvolvido pelo grupo de pesquisa do LEDES/DCT/UFMS como sistema gerenciador de conteúdo para aplicações Web. Um sistema gerenciador de conteúdo (*Content Management System* - CMS) têm por

finalidade separar o gerenciamento das páginas de conteúdo do projeto de interface (design gráfico). O design do layout das páginas é definido utilizando modelos (*templates*), enquanto o conteúdo é armazenado em banco de dados ou em arquivos textos independentes do *template*. Quando um usuário solicita o carregamento de uma página, as partes são combinadas para produzirem a página HTML padrão. A página resultante pode incluir conteúdos de diferentes fontes.

Segundo Pereira e Bax [Pereira e Bax 2002], o processo subjacente dos CMSs é organizado em três etapas básicas: criação, gestão e publicação. O CMS tem como objetivo permitir que os próprios usuários colaboradores, no papel de autores, alimentem o sistema com novos dados sem a necessidade de intermediários ou usuários com conhecimento de tecnologia. Em seguida, estes dados são armazenados em repositórios para serem tratados (gerenciados, padronizados, formatados e publicados) pelo CMS. O CMS deve gerir também as revisões, atualizações e o controle de acesso, garantindo confiabilidade ao que é publicado e segurança quanto à propriedade e a autoria dos dados.

Para a especificação do *framework* Titan foi utilizada a metodologia de desenvolvimento de projetos dirigidos por exemplos proposta por Johnson [Johnson 1993]. Por meio da revisão de uma série de sistemas de gerenciadores de conteúdo anteriormente implementados pôde-se especificar o *framework* Titan, que permite especificar uma arquitetura de software que recebe como entrada uma linguagem de marcação estendida (*Extensible Markup Language* - XML) e a transforma, em tempo de execução, em um gerenciador de conteúdo. A arquitetura do Titan possui como característica ser caixa-branca, flexível e estensível para outros domínios. Desta forma, se uma determinada funcionalidade não pode ser instanciada pelo *framework* é possível criar um novo componente para a respectiva funcionalidade. A linguagem genérica de entrada se encarrega de garantir que a configuração deste novo componente possa ser realizada como nos demais e que funcione de maneira harmônica com os outros componentes.

Neste artigo é apresentado o Titan estendido e adaptado para atender ao domínio. O desenvolvimento do ambiente de desenvolvimento envolveu, portanto, a extensão do

*framework* Titan, criando-se novos componentes e tipos para englobar os padrões do domínio de educação a distância. Por ser um *framework* do tipo caixa-branca este processo torna-se possível e sustentável. Para tanto foi realizado um mapeamento entre os padrões obtidos por meio da análise de sistemas no domínio e as funcionalidades já existentes no *framework*. Após estendido para este domínio, foram realizadas duas implementações de instâncias do Titan, utilizando os novos padrões e criando novos componentes para atender ao domínio de aplicações de educação a distância. Estas instâncias são sistemas reais, que atualmente estão em produção, e serão apresentadas na Seção 4.

A arquitetura do *framework* é formada por um núcleo (*core*) e um repositório, conforme ilustrada na Figura 1. O núcleo tem como característica ser imutável, indiferente da configuração da instância no domínio. Ele é responsável por receber como entrada os arquivos de configuração da instância (XML e SQL) e gerar uma aplicação em tempo de execução. Assim, apenas um núcleo é necessário para executar todas as aplicações instanciadas pelo *framework* em um mesmo servidor. O núcleo contém características e funcionalidades, tais como um sistema de autenticação e segurança, log, controle de reviões e autoria, que são descritos nesta seção. Tais características são automaticamente herdadas por todas as instâncias do *framework*.

Além do núcleo existe um outro conjunto de código, que não precisa ser replicado, denominado repositório (*repository*), responsável por armazenar as classes que podem ser estendidas do *framework*. Estas classes estão organizadas de acordo com o contexto em que estão inseridas: tipos e componentes.

Assim, o núcleo e o repositório agregam todos os artefatos responsáveis pela renderização da aplicação, que acontece em tempo-real por meio dos arquivos XML de entrada que o parametrizam. Na instância fica também um único arquivo de execução que efetua chamadas ao núcleo. Após a execução deste arquivo, ele irá buscar o arquivo principal de configuração, onde estão os caminhos para o núcleo e para o repositório, e irá incluir os arquivos de ambos de acordo com a necessidade.

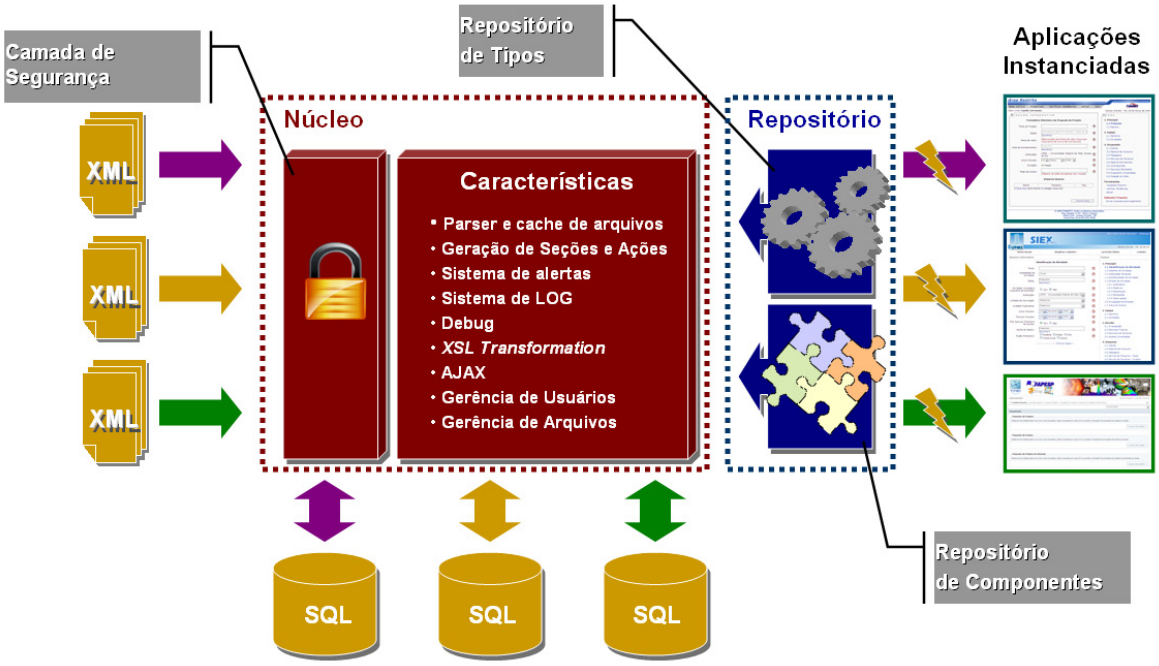


Figura 1. Arquitetura do *framework* Titan.

O Titan foi construído com uso de tecnologias de Engenharia de Software Livre (*Open Source*) [Dasilveira e Cassino 2003] e fez uso de técnicas e ferramentas colaborativas, baseadas no conceito de desenvolvimento de comunidades de software livre. A linguagem de programação utilizada é o PHP [Converse e Park 2003] e optou-se por utilizar a linguagem de marcação XML para configuração das instâncias. Assim, todas as instâncias no servidor usam o mesmo conjunto de código de execução (arquivos PHP) localizado na mesma pasta do servidor e cada instância tem uma pasta separada com suas específicas configurações (arquivos XML). Além de diminuir o espaço em disco no servidor eliminando repetição de código, esta estratégia permite que uma atualização do código do *framework* afeta ao mesmo tempo todas as instâncias no servidor, independente da sua configuração. Todo o código do Titan está disponível em um repositório público com controle de versões. Desta forma, além de se obter o código fonte, é possível atualizar versões do *framework* com um único comando do controlador de versões.

Dentre os artefatos de software reutiliáveis e extensíveis do *framework*, os Tipos (*types*) são classes responsáveis por tratar os tipos de dados no *framework*. Quando um formulário é criado ou é necessário carregar um dado da base de dados, este deve ser convertido para um objeto do *framework* e em seguida o Titan consegue manipular seus diferentes usos, tais como, em formulários HTML, arquivos PDF, cálculos e validação. Por exemplo, se o usuário deseja gerenciar um dado do tipo DATA (mais tecnicamente, um tipo *timestamp*), o *framework* converte este tipo para um objeto da classe *Date*, aplicando conversores de tipos e máscaras necessárias. Após ter sido carregado o respectivo tipo pode-se utilizá-lo em diferentes situações na aplicação, por exemplo, no cálculo de períodos de tempo ou idade.

O outro conjunto de artefatos retutilizável e extensível são os componentes. Uma WebApp possui vários propósitos ou categorias (Informativas, Interativas, Transacionais, *Workflow*, Colaborativas, Comunidades on-line e Portais). Uma seção da WebApp é definida como uma área específica em função de seu contexto. Por exemplo, alguma seções existentes em vários portais são: notícias, eventos, enquete, fale conosco e contato.

Para gerenciar cada uma das seções da aplicação existe uma entidade seção no *framework* Titan. Por exemplo, uma possível área de notícias do website é gerenciada por uma seção no CMS, enquanto uma área de agenda e eventos é gerenciada por outra seção específica para esse fim. Cada seção da instância possui um conjunto de arquivos de configuração próprio e faz, obrigatoriamente, uso de um componente do *framework*. Os componentes são, portanto, artefatos de código com funcionalidades específicas e que, quando parametrizados, geram seções.

Os componentes do Titan são armazenados no Repositório de Componentes e quando os componentes existentes não conseguem atender a demanda de determinada funcionalidade do website podem ser criados novos componentes ou estendidos os existentes. Várias seções da instância podem utilizar ao mesmo tempo o mesmo componente, isto ocorre quando se deseja tratar de forma semelhante seções distintas onde

mudam-se apenas campos ou rótulos. Desta forma, modificando-se os parâmetros de entrada dos componentes têm-se diferentes seções.

Exemplificando, o componente *global.generic* do Titan permite efetuar a listagem, criação, edição e remoção de itens de uma seção. Neste componente estão embutidas uma série de funcionalidades, tais como, busca, ordenação, paginação, validação, impressão de relatórios quantitativos com gráficos, formulários com múltiplos passos, RSS, entre outras. Assim, este único componente consegue atender as diferentes seções. No entanto, quando uma seção que possui funcionalidades muito distintas, como por exemplo uma seção de enquetes, necessita ser gerenciada, faz-se necessário criar um novo conjunto que artefatos que possibilite a instanciação destas funcionalidades.

Cada seção do CMS é formada por uma coleção de ações. Cada ação é uma funcionalidade bem específica da seção. Por exemplo, a listagem de itens de uma seção é uma ação, enquanto a criação de novos itens ou edição de itens existentes são outras ações. Cada ação de uma seção faz uso de um motor (*engine*), que fisicamente é um conjunto de três arquivos que renderizam as ações e têm funções bem definidas: preparação (*prepare*), visualização (*view*) e submissão (*commit*). Os arquivos de preparação separam a camada de execução, persistência e acesso ao SGBD da camada de visualização da ação. O arquivo de submissão tem como finalidade capturar e salvar as modificações e interações do usuário com a aplicação. Várias ações da seção podem fazer uso de um único motor.

O *framework* Titan possui uma série de características e funcionalidades embutidas em sua API (*Application Programming Interface*) que permitem ao usuário criar gerenciadores de conteúdo baseados no paradigma Web 2.0 [Oreilly 2005]. Considerando apenas o domínio de educação a distância, as características mais relevantes são apresentadas a seguir:

a. *Parser* e cache de arquivos de configuração: O *framework* efetua o parser dos arquivos de entrada XML, validando a estrutura por meio de arquivos DTD (*Document Type Definition*) e transformando as informações de configuração da instância em linguagem de máquina (*bytecode*). Para otimizar este processo o

*framework* prepara os arquivos XML transformando-os de XML para a sintaxe PHP. Isto otimiza o desempenho do sistemas por utilizar o parser nativo do interpretador da linguagem de programação.

b. Controle de acesso e autenticação: O Titan possui uma estrutura de controle de acesso baseada em usuários, tipos e grupos. Um usuário do *framework* é uma entidade que representa um usuário real da instância. Esta entidade possui um *login* único no sistema, um e-mail e uma senha criptografada de forma irreversível [W3C 1998]. Estas configurações são determinadas pelo tipo do usuário, que define a forma de cadastro de novos usuários daquele tipo, a quais grupos o usuário estará vinculado quando seu cadastro for realizado e quais os campos de dados estarão vinculados ao seu cadastro. Há três formas de cadastro possíveis:

    - Pública (*public*): Qualquer usuário pode se cadastrar publicamente para ter acesso à instância;

    - Protegida (*protected*): Qualquer usuário pode se cadastrar publicamente, mas requer que outro usuário, com as devidas permissões, habilite seu cadastro para que tenha acesso à instância; e

    - Privado (*private*): O formulário de cadastro somente pode ser acessado por usuários da instância com as devidas permissões.

    A configuração de tipos de usuários também pode definir se o tipo será autenticado por um serviço de diretórios pelo protocolo LDAP (*Lightweight Directory Access Protocol*) possibilitando que a instância se integre a bases de usuários pré-existentes ou que sirva de ferramenta gestora para estas bases. Em sistemas integrados esta base única de autenticação é fundamental. É fácil perceber a importância deste recurso no domínio de educação a distância, onde se têm diversos sistemas distintos que os alunos podem acessar e não se deseja uma uma coleção de logins e senhas que eles possam facilmente esquecer.

O controle de acesso as seções e ações da aplicação instanciada é definido pelas permissões do usuário. As permissões de acesso, no entanto, são vinculadas a grupos e não diretamente a usuários. Cada usuário herda estas permissões dos grupos aos quais está vinculado. Para ter acesso a instância, cada usuário deve estar vinculado a pelo menos um grupo de usuários. Permissões podem ser vinculadas e desvinculadas aos grupos em qualquer momento, afetando todos os usuários do grupo.

Este controle de usuários possibilita que instâncias do framework funcionem como grandes comunidades, onde cada usuário possui sua própria área restrita, sua produção pode ser auditada pelo registro de autoria e as permissões e visões sobre os dados podem variar em função do grupo ao qual o usuário, tal como, por exemplo, um grupo de professores e outro de alunos.

c. Busca, paginação e ordenação: A estrutura de classes do *framework* foi implementada para atender o conceito CRUD (*Create, Read, Update and Delete*) de armazenagem persistente (*persistent storage*) [Kilov 1998]. Desta forma, cada componente implementado oferece uma maneira simples e usual de criar, visualizar, editar e apagar itens de suas seções. Além disso, o *framework* oferece, para cada componente baseado neste conceito, a busca, paginação e ordenação dos itens já criados.

d. Gráficos e relatórios quantitativos: Para cada componente CRUD é possível habilitar a geração automática de relatórios quantitativos, que são mostrados pela instância na forma de gráficos, conforme Figura 2. Os gráficos são gerados levando em consideração os campos dos formulários. Novamente, uma ferramenta importante no domínio de educação a distância, permitindo resgatar e acompanhar visualmente os resultados de atividades, participação e o evolução dos alunos.

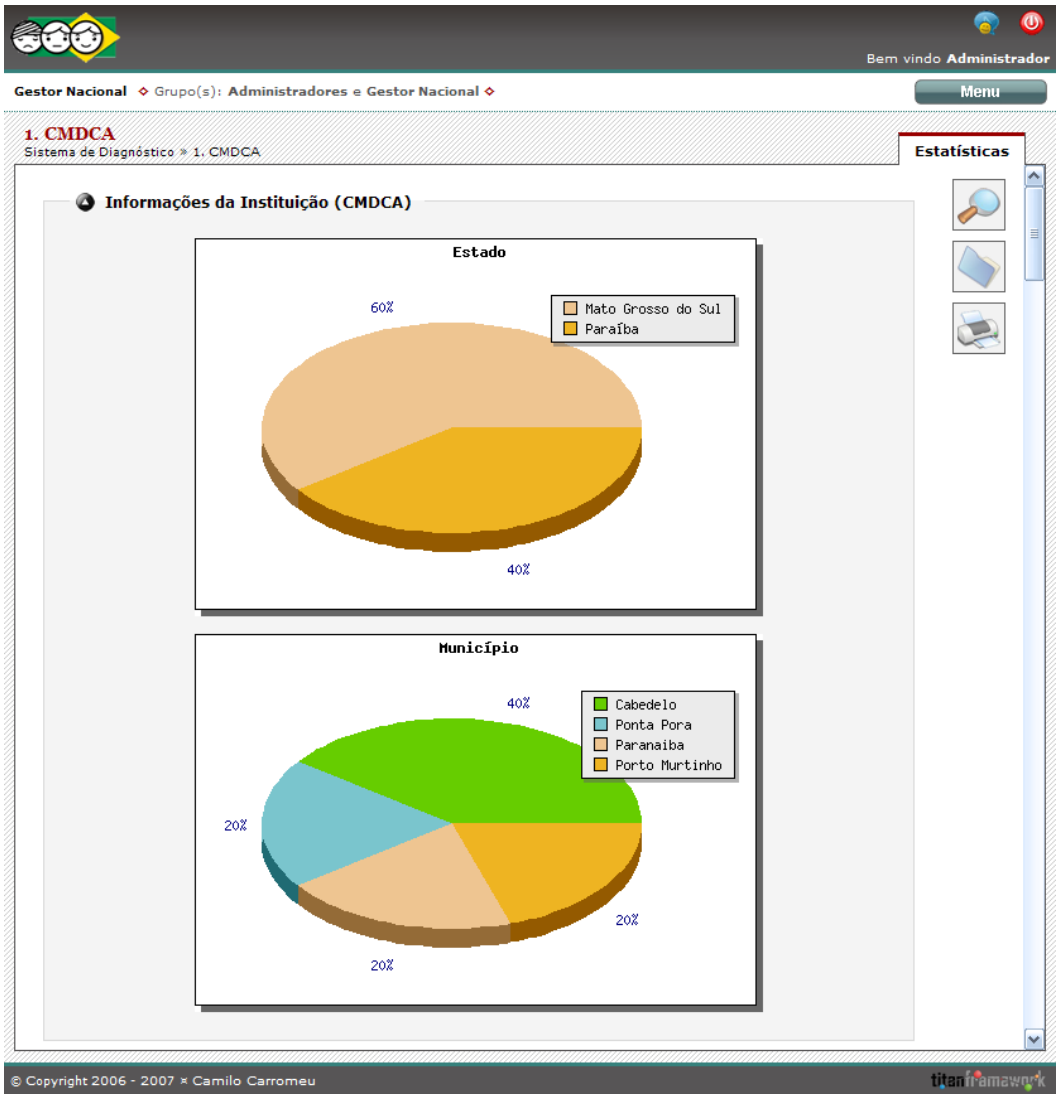


Figura 2: Tela dos relatórios quantitativos de uma instância do *framework* Titan.

e. Monitoramento de modificações: Outra funcionalidade implícita às seções geradas por componentes CRUD é a possibilidade de acompanhar os últimos itens inseridos e modificados por meio da utilização de links RSS (*Really Simple Syndication*). Desta forma, o usuário pode utilizar agregadores RSS externos para acompanhar as modificações nas seções sem que seja necessário efetuar *login* na ferramenta. Se novas atividades são inseridas, por exemplo, os alunos que monitoram a seção específica podem tomar conhecimento imediatamente por meio de seus

agregadores, sem a necessidade de estarem acessando a ferramenta ou mesmo sua caixa de e-mail.

f. Ajax (*Asynchronous Javascript And XML*): Ajax é uma técnica de desenvolvimento Web utilizada para criar WebApps interativas [Garrett 2005]. Sua utilização no *framework* Titan contribui para manipular dados entre o cliente e o servidor de forma invisível para o usuário. O uso desta técnica faz com que parte da carga de processamento da instância seja distribuída com o cliente, não sobrecarregando o servidor de atividades de renderização desnecessárias de páginas e evitando o recarregamento da página no cliente. O intuito é simular a experiência de uma aplicação desktop.

g. Controle de versões e auditoria: Pode ser ativada nas seções de componentes CRUD da instância do *framework* o suporte a auditoria e controle de versões. Quando este recurso está habilitado na seção, o *framework* grava automaticamente todas as modificações realizadas na seção. Isto inclui criação de novos itens, edição e remoção de itens existentes. Juntamente com cada modificação gravada é armazenado o autor da modificação e a data-hora em que ocorreu tal modificação. Isto impede por completo a perda de dados da aplicação como, por exemplo, no caso de remoção de itens por acidente ou invasão do sistema.

h. Editor de páginas ricas: O Titan possui um editor integrado de páginas ricas, ou seja, textos com vínculo de imagens e arquivos, formatação e tabelas. Além disso, em qualquer momento arquivos enviados podem ser reutilizados pelo autor, evitando o reenvio e o gasto desnecessário de espaço em disco no servidor.

i. Geração automática para formato de impressão e PDF: A visualização de itens das seções de componentes CRUD oferece, automaticamente, a visualização para impressão do item, removendo cabeçalhos, rodapé e outros itens desnecessários que aumentam o gasto de fita, tinta ou toner da impressora. Também oferece a geração em arquivo PDF dos itens, assim qualquer texto produzido pela aplicação pode ser salvo no computador do usuário para posterior visualização.

j. Sistema de alertas: O *framework* oferece um sistema de alertas por e-mail. A configuração deste sistema é realizada na instância vinculando o monitoramento das ações que desejar a grupos de usuários. Assim, qualquer usuário gestor da aplicação pode configurar quais grupos receberão quais alertas. Os alertas podem ser disparados sempre que um novo item é criado, quando alguma informação for modificada ou removida. Quando uma ação vinculada for ativada (como, por exemplo, a modificação de um item) os usuários vinculados àquele grupo recebem um alerta por e-mail.

k. Sistema de *debug* e *log*: A API do *framework* foi implementada com o conceito de exceções (*exception handling*). Desta forma, erros são capturados e mostrados ao usuário de forma "amigável". Além disso, pode-se configurar a instância para exibir mensagens de erro com informações técnicas ou não (*debug mode*), possibilitando que o sistema reaja de forma diferente durante o processo de instanciação e de produção da aplicação. Caso o sistema esteja configurado para não exibir erros com informações técnicas ele automaticamente armazena tais informações em arquivos de *log*, mantendo registrada a ocorrência e possibilitando a consulta pela equipe técnica responsável.

l. Salas de Chat: Uma instância do *framework* pode ser configurada para habilitar salas de chat (do inglês *chatroom*) para propiciar a conferência síncrona por meio da troca de mensagens entre os usuários da aplicação. Ferramentas de chat são amplamente utilizadas em cursos de educação a distância por possibilitarem ao professor simular uma sala de aula virtual. No Titan, cada grupo de permissões criado no controle de acesso pode ser configurado com uma sala de chat própria. Desta forma, todos os usuários de um grupo específico podem utilizar a sala deste grupo, de acesso restrito a usuários de outros grupos, assim pode-se criar diferentes turmas. Além disso, todos os usuários possuem acesso a uma sala denominada "Geral", e se o usuário receber mensagens enquanto o chat estiver fechado ele é avisado por meio de um alerta intermitente, possibilitando que o usuário navegue pela ferramenta ao mesmo tempo em que utiliza o chat.

m. *XSL Transformation*: A interface de uma instância do *framework* pode ser modificada usando as transformações XSLT (*Extensible Stylesheet Language Transformation*), padrão global para criação de *templates*. Toda a saída resultante pode ser convertida para XML e, então, folhas de estilo e arquivos XSL podem ser aplicados para modificar a visualização da instância. Quando a mudança na interface não é necessária este recurso pode ser desabilitado para otimizar o desempenho.

As características implícitas do Titan demonstram sua aptidão para a geração de aplicações Web no domínio. Somando-se a isso o *framework* foi extendido, tendo-se criados novos tipos e componentes específicos para atender à especificidades dos sistemas de educação a distância. Na próxima sessão serão apresentados dois estudos de caso e o seu desenvolvimento.

## 4. Estudos de Caso

O uso do ambiente e do *framework* proposto permite uma grande economia de recursos durante a fase de implementação de novos membros para a família de produtos de software do domínio de educação a distância. Uma grande quantidade de requisitos não-funcionais, tal como o sistema de usuários e controle de acesso, são implícitos às instâncias, não sendo necessário sua implementação. Assim, programadores podem ser alocados em regras de negócio e requisitos funcionais do sistema.

Entretanto, é necessário observar que um *framework* pode criar barreiras em determinadas situações, ou seja, os pontos-fixos de sua arquitetura podem originar funcionalidades muito diferentes das desejadas para a aplicação alvo. Por esta razão serão apresentados dois estudos de caso para validar o ambiente proposto no domínio de educação a distância. Ambos são aplicações reais que estão em uso atual.

## 4.1 Sistema de Informação Acadêmico a Distância – SIAD

O SIAD têm por objetivo atuar como uma ferramenta de gestão de cursos de educação a distância. Por meio deste sistema pode-se gerenciar a matrícula de novos alunos, separá-los em turmas, lançar presença e notas de avaliações de cada aluno, elaborar a grade e ementa do curso, o plano de aula, o diário de classe e emitir relatórios.

Alunos, professores e funcionários da secretaria do curso são atores do sistema. O SIAD oferece visões diferentes dos dados para cada um destes. Desta forma, a um aluno que acessa o sistema será permitida a visualização apenas dos dados que lhe dizem respeito. Da mesma forma, um professor acessando o sistema têm possibilidade de visualizar e gerenciar dados de sua matéria e alunos pertencentes às suas turmas.

Como o SIAD é um sistema bastante complexo, será descrito suscintamente suas funcionalidades e as modificações realizadas no Titan para estendê-lo. A primeira mudança que merece destaque foi a necessidade de efetuar logon na ferramenta através do CPF do usuário. Assim, um usuário que fosse inscrito no sistema jamais precisaria realizar um novo cadastro.

Pode-se imaginar uma situação em que têm-se um usuário que, no passado, foi aluno de um curso de educação a distância e, portanto, esta cadastrado no sistema como um tipo ‘Aluno’. Suponhamos então que este usuário se torna um professor de um outro curso a distância. O sistema deve possibilitar que este usuário seja também cadastrado com o tipo ‘Professor’.

Para atender a este requisito foi criado um novo componente do Titan, denominado “*ead.user*”. No momento do cadastro de um novo usuário no sistema deve-se inserir, primeiramente, o CPF da pessoa. Com isso o sistema pode verificar se essa pessoa já esta cadastrada no sistema. Se não estiver, então um cadastro completo do novo usuário pode ser efetuado. Se já estiver então apenas os dados do papel novo serão requisitados. Utilizando o exemplo citado acima, se a pessoa tinha um cadastro de ‘Aluno’ os dados

existentes não são suficientes para suprir o novo papel de ‘Professor’, assim, dados específicos deste tipo serão requisitados.

Outros componentes implementados para estender o *framework* no domínio de educação a distância foram:

a. *ead.avaliation*: Este componente é responsável por implementar o subsistema de avaliação de notas, onde avaliações, como provas e atividades, são configuradas;

b. *ead.course*: Este componente implementa uma seção de cursos, com um recurso especial que permite clonar um curso inteiro com apenas um clique;

c. *ead.discipline*: Este componente permite que seja gerado um código, em um padrão específico, para novas disciplinas;

d. *ead.frequency*: Componente que possibilita, de forma simples e visual, o lançamento de presenças e faltas para os alunos. A tela de lançamento relaciona a turma com o aluno, conforme pode ser visualizado na Figura 3;

e. *ead.oferta*: Vínculo da estrutura curricular a um determinado curso de um pólo;

f. *ead.plan*: A seção instanciada através deste componente permite a criação do plano de ensino pelo professor;

g. *ead.registration*: Responsável pela seção de matrículas, onde os alunos são inseridos às turmas;

h. *ead.report*: Geração de relatórios e extração de dados e indicadores do sistema;

i. *ead.scores*: Implementa a seção de lançamento de notas;

j. *ead.struct*: Este componente possibilita o vínculo de disciplinas com cursos, criando a estrutura curricular; e

k. *ead.turma*: Possibilita a criação de turmas e o vínculo de alunos a elas.

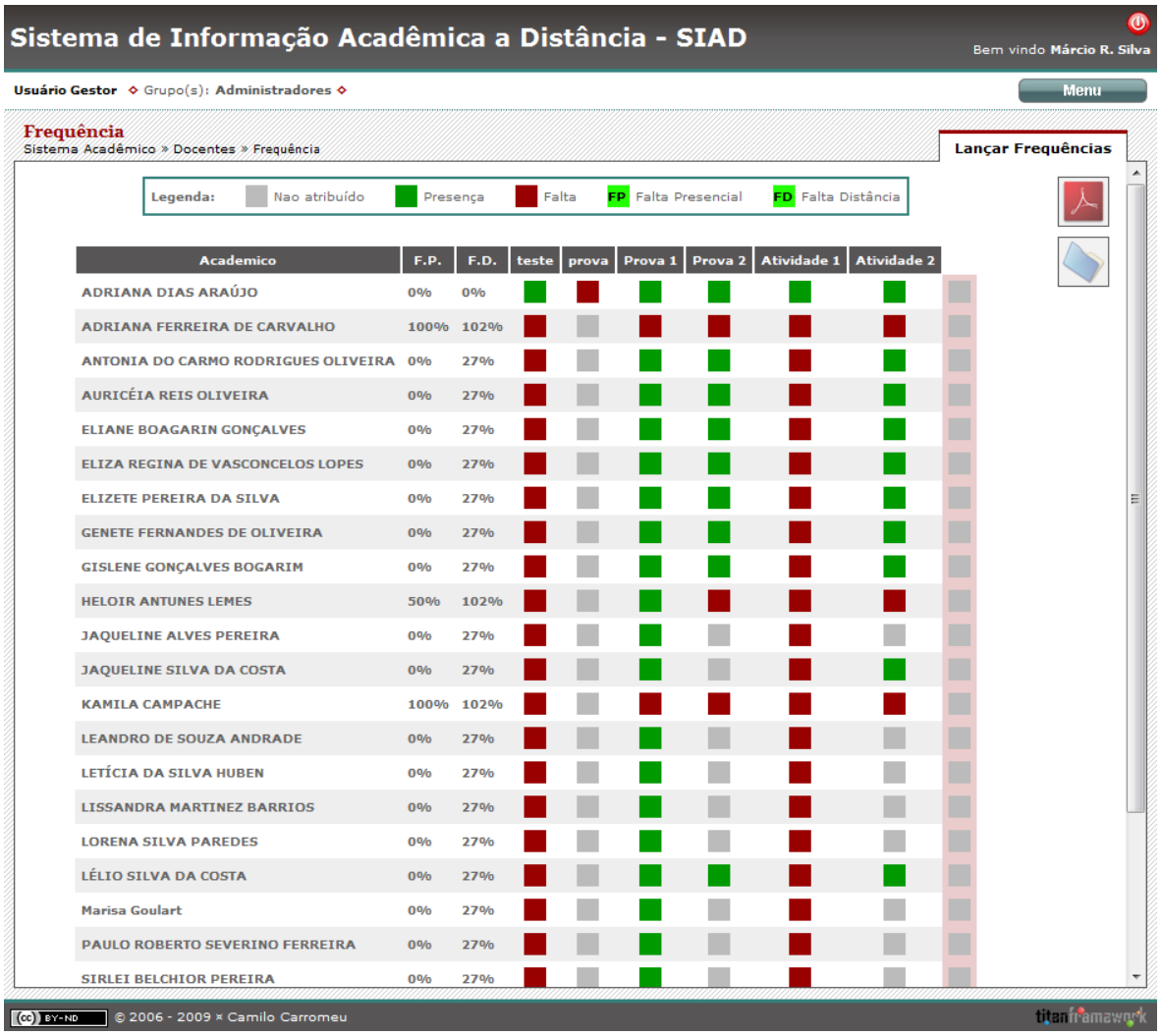


Figura 3: Seção “Frequência”, derivada do componente “*ead.frequency*”.

Além de estender os componentes, no SIAD fez-se necessário estender alguns tipos do Titan para atender a todos os requisitos. O framework, novamente, permite que isto seja feito de forma rápida e fácil. Assim, foram criados os seguintes tipos:

a. *siad.Avaliation*: Têm por finalidade prover um campo para os formulários do sistema que carregue dados de avaliações;

b. *siad.Departamento*: Provê um campo que lista os departamentos da instituição registrados no banco de dados da aplicação; e

c. *siad.Year*: Campo que força a entrada de valores de 4 dígitos, representando um ano.

Com a implementação de componentes e tipos derivados dos originais do Titan, podê-se estender o *framework* de forma que atendesse a todos os requisitos do SIAD. Com isso gastou-se recursos apenas para a implementação de regras de negócio próprias, sem a necessidade, em nenhum momento, de repetição de código. Estes fatores permitiram que o sistema fosse implementado em um tempo muito inferior ao que normalmente levaria.

## 4.1 Sistema de Cursos da Escola de Conselhos

O Programa Escola de Conselhos é uma ação de extensão da UFMS que desde de abril de 2005 realiza inúmeras atividades voltadas ao aperfeiçoamento e qualificação da participação da sociedade na definição e controle das políticas públicas de atendimento e defesa dos direitos humanos e da cidadania, em especial dos atores que compõem a rede de defesa dos direitos da criança e do adolescente.

Devido ao seu caráter de oferecer qualificação, a Escola de Conselhos iniciou em 2008, junto ao LEDES, o desenvolvimento de uma aplicação Web para educação a distância objetivando, assim, atender aos inúmeros alunos, geograficamente distantes, de seus cursos. A principal função era a de disponibilizar ferramentas pelas quais os alunos pudessem acompanhar as aulas, obter material didático, submeter atividades realizadas e oferecer encontros virtuais através de uma ferramenta de *chat*.

Novamente, foi utilizado o Titan *Framework* para a implementação. Uma de suas características é uma ferramenta embutida de salas de *chat*. Desta forma, através de reuso, foram atendidos todos os requisitos não-funcionais e um dos principais requisitos. Para atender aos demais foram criados componentes para estender o *framework*:

a. *course.library*: Implementa a seção de biblioteca do sistema, onde os alunos podem fazer download de material didático;

b. *course.task*: Possibilita a instanciação da seção de atividades. Através desta seção os professores podem postar o enunciado de questões que são respondidas pelos alunos. Cada aluno pode visualizar apenas sua produção e o professor, ao final de um prazo pré-estabelecido, pode corrigir todas as respostas; e

c. *course.user*: Este componente foi especialmente criado para possibilitar a matrícula *online* em cursos com inscrições abertas.

Por ser um sistema mais simples que o SIAD, o Sistema de Cursos da Escola de Conselhos não precisou de novos tipos e a quantidade de novos componentes foi mais reduzida. Assim mesmo, ofereceu grandes desafios já que o público alvo da ferramenta era, muitas vezes, leigo no uso do computador. Este sistema foi de grande valia no teste e validação de usabilidade de instâncias do Titan.

## 5. Conclusão

Por meio das ponderações apresentadas e dos resultados obtidos na implementação dos casos de uso do SIAD e do Sistema de Cursos da Escola de Conselhos, pode-se concluir que o Titan é uma excelente solução na implementação de WebApps no domínio de educação a distância. Através da utilização de seu conjunto de código genérico é possível a simples parametrização para obter-se uma grande quantidade de funcionalidades. Os requisitos não atendidos por estes artefatos de software podem ser alcançados através da adaptação de seus componentes e tipos, uma tarefa prevista, modularizada e onde todo o esforço é aplicado na implementação de regras de negócio específicas do sistema.

A principal contribuição deste trabalho foi apresentar um novo ambiente para produção de software de qualidade no domínio de educação a distância. O Titan é uma solução baseada em software livre, suas instâncias podem executar sobre plataformas totalmente abertas, como sistema operacional Linux, banco de dados PostgreSQL e servidor Web Apache com PHP.

Além dos estudos de caso apresentados, o *framework* Titan, validado no domínio desta pesquisa, está sendo utilizado na implementação de novos sistemas Web e projetos de

pesquisa, tais como: Projeto WebPIDE - Plataforma Aberta de Integração e Avaliação de Dados Educacionais - CAPES/INEP (http://webpide.ledes.net); CISM - Central de Indicadores e Serviços Municipais - Fundect (http://cism.ledes.net); portal do PAIR – Programa de Ações Integradas e Referênciais de Enfrentamento à Violência Sexual Infanto-Juvenil no Território Brasileiro - SEDH da Presidência da República (http://pair.ledes.net); e-SAPI / Portal da Carne - CNPq, Embrapa e MAPA (http://e-sapi.ledes.net); portal de Boas Práticas Agropecuárias - CNPq, Embrapa e MAPA (http://bpa.ledes.net); portal do LEDES (http://www.ledes.net); e, portais de departamentos e Pró-Reitorias da UFMS, por exemplo, do Departamento de Computação e Estatística (http://www.dct.ufms.br) e da Pró-Reitoria de Extensão, Cultura e Assuntos Estudantis (http://www.preae.ufms.br).

## 8. Referências Bibliográficas


Carromeu, C. *Linha de Produtos de Software no Processo de Geração de Sistemas Web de Apoio à Gestão de Fomento de Projetos*. Dissertação de Mestrado em Ciência da Computação. Departamento de Computação e Estatística. Universidade Federal de Mato Grosso do Sul. Campo Grande, MS, Brasil. Novembro de 2007.

Selfe, C; Eilola, J. *The tie that binds: building discourse communities and group cohesion through computer-based conferences*. Collegiate Microcomputer, Terre Haute (IN), 64, p.339-348,1988.

Bates, A.W. *Interactivity as a criterion for media selection in distance education*. In: The Asian Association of Open Universities 1990 Annual Conference, Universitats Terbuka. Paper.

Seaton, W.J. *Computer mediated communication and student self-directed learning*. Open Learning, Essex (GB), v.8, n.2, p.49-54, 1993.

Nalley, R. *Designing compuler-mediated conferencing into instruction*. In: Berge, Z.L., COLLINS, M.P. (Eds.). Computer-mediated communication and the online classroom. Cresskill (NJ): Hampton Press, 1995. p. l1-23. v.2: Higher Education.

Burge, E.; Roberts, J.M. *Classrooms with a difference: a practical guide to the use of conferencing technologies*. Toronto: The Ontario Institute for Studies in Education, Distance Learning Office, 1993.

Jonassen, D.H. *Hypertext/hypermedia*. Englewood Cliffs (NJ): Educational Technology Publications, 1989.

Pressman, R. S. *Software Engineering: A Practitioner's Approach*. 6th ed. New York, USA: McGraw Hill, 2005.

Cheesman, J.; Daniels, J. *UML Components - A Simple Process for Specifying Component-based*. Addison-Wesley, 2001.

Bosch, J.; Bengtsson, P. O.; Molin, P.; Mattsson, M. *Object oriented framework - problems & experiences*. Wiley & Sons, 1999.

De Paez, R. A. *Un acercamiento a la reutilización en ingeniería de software*. EAFIT Magazine, Medellín, Colombia, n. 114, p. 45-63, 1999.

GIMENES, I. M. S.; TRAVASSOS, G. H. *O enfoque de linha de produto para desenvolvimento de software*. Evento integrante do XXII Congresso da SBC – SBC 2002. XXI Jornada de Atualização em Informática. Sociedade Brasilera de Computação, 2002.

Braga, R. T. V. *Um Processo para Construção e Instanciação de Frameworks baseados em uma Linguagem de Padrões para um Domínio Específico*. Instituto de Ciências Matemáticas e de Computação da USP-SC, São Carlos, São Paulo, Brasil, 2003.

Balzerani, L.; Di Ruscio, D.; Pierantonio, A.; De Angelis, G. *A Product Line Architecture for Web Applications*. ACM Symposium on Applied Computing, 2005.

Sommerville, I. *Software Engineering*. Addison-Wesley, 2001.

Johnson, R. E. *Documenting Frameworks Using Patterns*. OOPSLA'92: Conference proceedings on Object-oriented programming systems, languages, and applications. Vancouver, British Columbia, Canada: ACM Press, 1992.

Johnson, R. E. *How to design frameworks*. Tutorial at OOPSLA'93, 1993.

Fayad M. E.; Johnson, R. E.; Schmidt, D. C. *Software Patterns*. Communications of the ACM, ACM Press, v. 39, n. 10, p. 37-39, 1996.

Fayad, M. E.; Johnson, R. E.; Schmidt, D. C. *Building Application Frameworks: Object-Oriented Foundations of Framework Design*. Wiley & Sons, 1999.

Fayad, M. E.; Schmidt, D. C. *Object-Oriented Application Frameworks*. Communications of the ACM, ACM Press, New York, NY, USA, v. 40, n. 10, p. 32-38, 1997.

Pree, W. *Hot-Spot-Driven Development*. Fayad, M.; Johnson, R.; Schmidt, D. (Ed.). Building Application Frameworks: Object-Oriented Foundations of Framework Design. Wiley & Sons, 1999.

Braga, R. T. V.; Germano, F. S. R.; Masiero, P. C. *A Pattern Language for Business Resource Management*. Proceedings of the 6th Annual Conference on Pattern Languages of Programs (PLoP'99). Monticello, Illinois, USA, 1999.

Brugali, D.; Sycara, K. *Frameworks and Pattern Languages: An Intriguing Relationship*. ACM Computing Surveys, ACM Press, v. 32, n. 1, p. 2-7, 1999.

Frye, J.; Yoder, J. *The Hillside Group*. 2001. Disponível em: http://hillside.net [10/01/2009].

Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerland, P.; Stal, M. *Pattern-Oriented Software Architecture - A System of Patterns*. Wiley & Sons, 1996.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns: Abstraction and Reuse of Object-Oriented Design*. Lecture Notes in Computer Sciente, Springer-Verlag, Berlin, Alemanha, n. 707, p. 406-431, 1993.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

Fayad, M. E.; Schimidt, D. C.; Johnson, R. E. *Domain-Specific Application Frameworks*. Wiley Computer Publishing, 1999.

Aragón, C. R. *Processo de Desenvolvimento de uma Linha de Produtos para Sistemas de Gestão de Bibliotecas*. Universidade Federal do Mato Grosso do Sul, Campo Grande, MS, Brasil, 2004.

D'Souza, D. F.; Wills, A. W. *Objects, Components and Frameworks with UML - The Catalysis Approach*. Addison-Wesley, 1999.

Crnkovic, I.; Hnich, B.; Jonsson, T.; Kiziltan, Z. *Specification, Implementation, and Deployment of Components*. Communications of the ACM, ACM Press, New York, NY, USA, v. 45, n. 10, p. 35-40, 2002.

França, L. P. A.; Staa, A. V. *Geradores de artefatos: Implementação e instanciação de frameworks*. Anais do XV SBES-2001 - Simpósio Brasileiro de Engenharia de Software. Rio de Janeiro, Brasil, 2001.

Batory, D.; Smaragdakis, Y. *Application Generators*, 2003. Disponível em: http://www.cs.umass.edu/~yannis [08/01/2009].

Clements, P. C.; Northrop, L. *Software Product Lines: Practice and Patterns*. Addison-Wesley, 2001.

Lai, C. T. R.; Weiss, D. M. *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison-Wesley, 1999.

Bosch, J. *Software Product Lines: Organizational Alternatives*. Proceedings of the 23rd international conference on Software engineering. Toronto, Ontario, Canada: IEEE Computer Society, 2001.

Kiczales, G.; Lamping, J.; Meddhekar, A.; Maeda, A.; Lopes, C. V.; Loingtier, J. M.; Irwin, J. *Aspect-Oriented Programming*. In proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS, 1997.

Tarr, P.; Ossher, H. *Multi-Dimensional Separation of Concerns and the Hyperspace Approach*. Proceedings Architectures and Component Technology: The State-of-the-Art in Software Development, 2000.

Frakes, W.; Terry, C. *Software Reuse: Metrics and Models*. ACM Computing Surveys (CSUR), ACM Press, v. 28, n. 2, p. 415-435, 1996.

Coplien, J. O. *Software Design Patterns: Common Questions and Answers*. The Patterns Handbook: Techniques, Strategies, and Applications. New York, NY, USA: Cambridge University Press, 1998.

Foote, B.; Johnson, R. E. *Designing Reusable Classes*. Journal of Object Oriented Programming, v. 1, n. 2, p. 22-35, 1988.

Shimabukuro, E. K. *Um Gerador de Aplicações Configurável*. Instituto de Ciências Matemáticas e de Computação, ICMC-USP, São Carlos, SP, Brasil, 2006.

Silva, R. P. *Suporte ao Desenvolvimento e Uso de Frameworks e Componentes*. Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brasil, 2000.

Pereira, J. C. L.; Bax, M. P. *Introdução à Gestão de Conteúdo*. Anais do III Workshop Brasileiro de Inteligência Competitiva e Gestão do Conhecimento, São Paulo, 2002.

Da Silveira, S. A.; Cassino, J. *Software Livre e Inclusão Digital*. Conrad Livros, 2003.

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

O'Reilly, T. *What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software*. 2005. O'Reilly Network, disponível em: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html [12/01/2009].

W3C. *SHA-1 - Secure Hash Algorithm*. 1998. Disponível em: http://www.w3.org/TR/1998/REC-DSig-label/SHA1-1_0 [12/01/2009].

Kilov, H. *Business Specifications: The Key to Successful Software Engineering*. Prentice Hall, 1998.

Garrett, J. J. *Ajax: A New Approach to Web Applications*. 2005. Disponível em: http://www.adaptivepath.com/ideas/essays/archives/000385.php [13/01/2009].