Documento de Arquitetura

Sistema de Automação de Formulários da Unichristus - FORMCHRISTUS

**Versão 01.00** 19 de maio de 2014

Histórico de Revisões

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Versão | Descrição | Autor |
| 29/04/2014 | 1.0 | Versão inicial do documento | Ari, Giancarlo, Glaydson |
|  |  |  |  |

**SUMÁRIO**

[1. Introdução 5](#_Toc388305490)

[2. Arquitetura da Aplicação 5](#_Toc388305491)

[2.1 Objetivos, Restrições Arquiteturais e Tecnológicas 5](#_Toc388305494)

[2.1.1 Plataforma 5](#_Toc388305495)

[2.1.2 Projeto de Iniciação Científica 5](#_Toc388305496)

[2.2 Descrição da arquitetura 6](#_Toc388305497)

[3. Visões Arquiteturais 6](#_Toc388305498)

[3.1 Visão de Casos de Uso 6](#_Toc388305500)

[3.1.1 Visão Geral 6](#_Toc388305501)

[3.1.2 Mecanismos Arquiteturais 7](#_Toc388305502)

[3.2 Visão Lógica 7](#_Toc388305503)

[3.2.1 Visão Geral 7](#_Toc388305504)

[3.3 Visão de Implantação 7](#_Toc388305505)

[3.4 Visão de Implementação 7](#_Toc388305506)

[3.4.1 Componentes Reutilizados 7](#_Toc388305507)

[3.5 Visão de Dados 7](#_Toc388305508)

[4. Padrões Adotados 8](#_Toc388305509)

[5. Referências 8](#_Toc388305510)

[6. Arquitetura 8](#_Toc388305511)

[6.1. Elementos que compõe a Arquitetura 8](#_Toc388305512)

[6.2. Modelos 9](#_Toc388305513)

[4 9](#_Toc388305514)

[6.3. Infraestrutura 9](#_Toc388305515)

[6.4. Domínio 10](#_Toc388305516)

[6.5. Visão 11](#_Toc388305517)

[Desenho GERAL da A rquitetura 12](#_Toc388305518)

[7. Padrões de Projeto 13](#_Toc388305519)

[8. Objetivos e Restrições Arquiteturais 13](#_Toc388305520)

[8.1. Requisitos básicos 13](#_Toc388305521)

[8.2. Estratégias de implementação 13](#_Toc388305522)

# Introdução

Neste documento iremos detalhar as principais partes da arquitetura proposta para o desenvolvimento do Sistema de Automação de Formulários Unichristus. A arquitetura é formada por diversos padrões de projeto, principalmente, padrões Orientados a Objetos com destaque no mercado. Iremos destacar em cada parte da arquitetura o motivo da sua criação e o qual a sua influência para a criação de sistemas de alta coesão, mas com baixo acoplamento.

O Documento de Arquitetura do Software provê uma visão geral da arquitetura, usando um conjunto de visões arquiteturais para tratar aspectos diferentes do software.

Este documento serve como um meio de comunicação entre o Arquiteto de Software e outros membros da equipe de projeto sobre as decisões significativas que forem tomadas durante o projeto.

# Arquitetura da Aplicação



## Objetivos, Restrições Arquiteturais e Tecnológicas

### Plataforma

O sistema FormChristus será desenvolvido na plataforma Java e executará em ambiente Web. A aplicação será implantada em servidor de aplicações GlassFish utilizando a infraestrutura já existente no ambiente do Centro Universitário Unichristus.

O sistema será autocontido, não possuindo quaisquer interações com outros sistemas já existentes na Unichristus, em particular o sistema “aluno online” que já é utilizado pela instituição para registros acadêmicos. Poderão ser carregados inicialmente para fins de teste, dados disponíveis em outros sistemas, mas sem a comunicação entre eles. O FormChristus possuirá sua própria base de dados, não estando também previstas quaisquer exportações de dados para sistemas externos.

### Projeto de Iniciação Científica

O sistema FormChristus é o resultado do projeto de Iniciação Científica desenvolvido na Unichristus no período 2013/2014, tendo como objetivo promover a automatização dos formulários que atualmente são preenchidos à mão dentro da instituição. O foco inicial do projeto é a automatização do preenchimento e geração dos formulários, embora outras funcionalidades possam ser acrescidas no futuro.

Para este projeto, foram selecionados pela coordenação do curso de Sistemas de Informação os quatro formulários abaixo:

* A1 – Programa de Atividade Complementar
* A3 – Relatório de Atividade Complementar
* A4 – Inscrição em Atividade Complementar
* E1 – Programa de Extensão

## Descrição da arquitetura

A arquitetura do sistema utiliza um modelo em camadas, composto de:

* Uma camada de **Apresentação**, que utiliza o Primefaces, uma implementação do framework Java Server Faces (JSF).
* Uma camada de **Negócio**, que utiliza o framework EJB e CDI (Context and Dependency Injection) para injeção de dependências e o framework Hibernate para mapeamento objeto-relacional.
* Uma camada de **Persistência**, que utiliza o Hibernate como framework de Mapeamento Objeto-Relacional, a biblioteca Ehcache para prover cache de segundo nível e injeção de dependência através do CDI (Context and Dependency Injection) para prover a conexão com o banco de dados juntamente com aplicação de um Pool de conexões implementado pelo servidor Glass Fish.
* Uma camada utilitária adicional para visualização e impressão dos formulários em seu formato real, em papel ou em arquivo de formato PDF. Para tal, utilizou-se os frameworks Jasper/IReports.

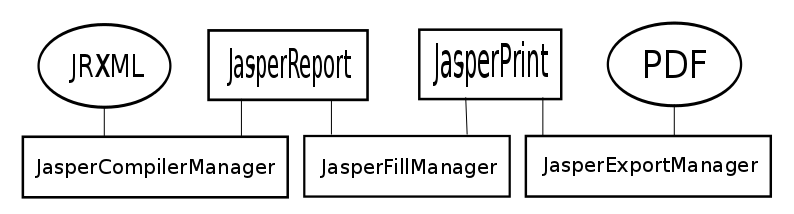
O *JasperReports* é um framework para a geração de relatórios. É uma ferramenta totalmente *open source* e gratuita, e a mais utilizada com esse propósito atualmente. Entre as funcionalidades do JasperReports podemos destacar:

* é capaz de exportar relatórios para diversos formatos diferentes, tais como PDF, HTML, XML, XLS, etc.
* aceita diversas formas de entrada de dados, tais como um arquivo XML ou CSV, conexão com o banco de dados, uma sessão do Hibernate, uma coleção de objetos em memória, etc.
* permite o uso de diagramas, gráficos, e até códigos de barras.

### Como o JasperReports funciona?

Um aspecto importante do JasperReportsé que o layout do relatório é definido em um arquivo XML, geralmente com a extensão .jrxml. Este XML possui todas as informações de formatação do relatório, e além disso, possui os campos que serão preenchidos posteriormente, de acordo com a fonte de dados utilizada (data source).Como dito anteriormente, a fonte de dados pode variar, e ser uma tabela em uma base de dados, ou ser um arquivo CSV, porém a formatação do relatório será a mesma em ambos os casos.

Os passos para gerar um relatório são bem simples. O primeiro passo é compilar o relatório em XML. Depois da compilação, o resultado é um objeto do tipo JasperReport. O próximo passo é preencher o relatório com os dados, e o resultado dessa etapa fica armazenado em um objeto do tipo JasperPrint. Esse objeto já representa o relatório finalizado, a partir dele podemos enviar para impressão diretamente, ou podemos exportar para um outro formato, tal como PDF por exemplo. Veja um diagrama ilustrando o processo



O JasperReports pode ser baixado [aqui](http://jasperforge.org/projects/jasperreports). O download inclui, além do jar do próprio JasperReports, vários outros jars do qual o JasperReports depende. Por exemplo, caso você utilize gráficos no seu relatório, precisará incluir ojfreechart no seu build path. Por padrão, qualquer projeto que utilize o JasperReportsprecisará incluir pelo menos as seguintes jars:

* commons-beanutils
* commons-collections
* commons-digester
* commons-logging
* groovy-all
* iText
* poi
* jasperreports (é claro)

### Para que serve o iReport?

O iReport é uma ferramenta desenvolvida pela mesma empresa do JasperReports, a JasperForge, e por isso é muito comum ver os dois sendo usados em conjunto. Uma das dificuldades ao trabalhar com os relatórios, está na definição do layout. É complicado escrever o layout totalmente em XML, sem ter que se aprofundar em todas as tags e atributos possíveis, e além disso posicionar todos os elementos corretamente. Na prática, é muito raro alguém editar o JRXML manualmente, e sim apenas para fazer alguns pequenos ajustes quando necessários. O processo normal é utilizar alguma ferramenta para gerar o JRXML automaticamente, e o iReport é utilizado com esse propósito.

O iReport é um aplicativo gráfico, que permite que você “desenhe” um relatório, utilizando uma palheta, e arrastando e soltando componentes, de forma bem parecida com a criação de interfaces e janelas  para programas. Ao salvar, automaticamente será gerado um JRXML que você poderá utilizar na aplicação que estiver desenvolvendo. A vantagem é que não é necessário que você conheça a fundo o XML a ser editado, economizando tempo de desenvolvimento. Ele também traz um conjunto pronto de templates que você já pode utilizar diretamente, ou então, escrever seus próprios templates e reaproveitá-los sempre que precisar criar um novo tipo de relatório.

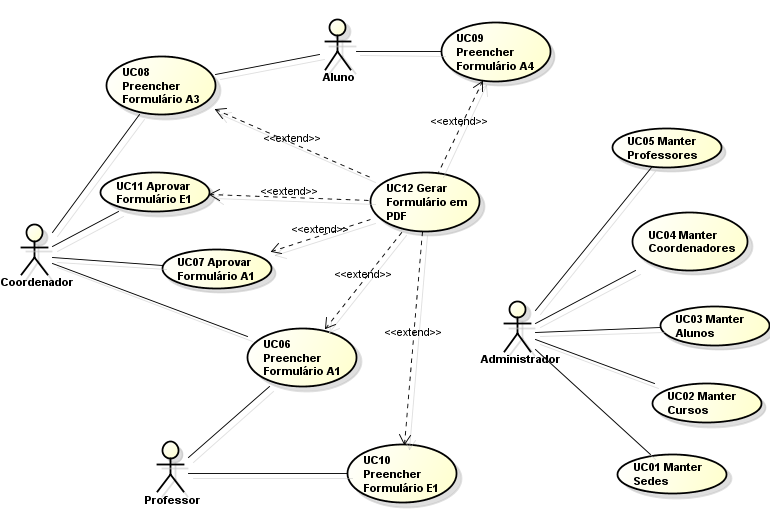
# Visões Arquiteturais



## Visão de Casos de Uso

### Visão Geral

A pretensão inicial da equipe do projeto, conforme solicitação da Unichristus, foi a automatização do preenchimento dos formulários, definindo-se assim as seguintes funcionalidades, apresentadas no Diagrama de Casos de Uso mostrado na Figura 1.



*Figura 1 – Diagrama de Casos de Uso*

Todos os casos de uso mostrados são significativos e foram implementados para este projeto. Os formulários escolhidos foram modelados usando a ferramenta IReports, o que gerou...

### Mecanismos Arquiteturais

Como exemplo de utilização da arquitetura descrita no item 2.1, apresentaremos dois casos de uso significativos

## Visão Lógica

### Visão Geral

## Visão de Implantação

A aplicação foi desenvolvida em dois módulos distintos, um contendo as camadas de negócio e persistência, e outro contendo a aplicação web propriamente dita. O módulo de negócio e persistência é compilado em um arquivo JAR que é utilizado como biblioteca para o módulo web. O arquivo resultante (EAR?) é publicado em um servidor de aplicações JEE (GlassFish). Esse módulo se comunica com um servidor de banco de dados PostGres também rodando no servidor.

## Visão de Implementação

A aplicação possui um único módulo web, publicado no servidor de aplicações conforme descrito no item anterior.

### Componentes Reutilizados

A seguir são listados os componentes utilizados pelo sistema – que são suas dependências. As bibliotecas estão todas disponibilizadas em arquivos JAR.

|  |  |
| --- | --- |
| **Nome do Componente** | **Versão** |
| Primefaces |  |
| Hibernate |  |
| Jasper |  |

## Visão de Dados

# Padrões Adotados

O Arquiteto de Software é o responsável por elaborar este documento e por manter a integridade do mesmo durante o processo de desenvolvimento do software. Ele deve:

* Aprovar todas as mudanças arquiteturais significativas e documentá-las.
* Fazer parte do comitê que decide sobre os problemas que tenham algum impacto arquitetural.

# Referências

1. <http://community.jaspersoft.com/project/jasperreports-server>
2. <http://community.jaspersoft.com/project/ireport-designer>
3. <http://mahtabrasheed.wordpress.com/2013/01/25/what-is-java-server-faces-jsf-and-primefaces/>
4. <http://www.devmedia.com.br/ejb-3-1-versus-cdi-revista-java-magazine-105/25040>
5. <http://www.mastertheboss.com/cdi/using-cdi-to-manage-ejb-dependencies>
6. <http://www.yaw.com.br/open/projetos/cdi-jsf-ejb3/>

# Arquitetura

O que é? E como é composta?

A arquitetura foi desenvolvida para ser totalmente de alta coesão e baixo acoplamento, e que ao mesmo tempo fosse independente de tecnologia de solução existentes no mercado.Utilizando o padrão MVC(Model View Controler).

* 1. Elementos que compõe a Arquitetura

Quais são os principais elementos da arquitetura?

A arquitetura é composta por alguns elementos, entenda-se classes, que em conjunto produzem o efeito desejado pela arquitetura como um produto final para o desenvolvimento.

Elementos pertencentes à arquitetura:

Modelo

Infraestrutura

Dominio

Visão

JSF

Nos próximos capítulos iremos discutir cada componente listado anteriormente e qual o seu papel dentro da arquitetura como um todo, além de discutir de forma sucinta a tecnologia e/ou o padrão adotado para a implementação do mesmo.

* 1. Modelos

O que é essa camada? E pra que serve?

No desenvolvimento de sistemas precisamos em muitas vezes fazer acesso a uma determinada base de dados. Em algumas linguagens de programação, principalmente as estruturadas, a separação real de funcionalidades na programação é muito complexa de forma que acessos a dados, consultas, entre outros elementos do sistema, acabam se misturando, o que ocasiona um alto acoplamento para termos alta coesão. Com a programação orientada a objetos isso já não ocorre em modelagens mais elaboradas. No caso da arquitetura desenvolvida temos uma camada apenas com classes que espelham o banco de dados ou camada de modelo. Nessa camada estão todas as classes mapeadas com as entidades do banco de dados.

# 

* 1. Infraestrutura

Qual a sua principal finalidade?

Quando estamos elaborando uma arquitetura para o desenvolvimento de sistemas, principalmente orientado a objetos, temos que nos preocupar com a separação real das camadas pertencentes à arquitetura. É nesse contexto que começamos a discutir os principais elementos da arquitetura, sendo que agora iremos começar detalhar a camada de infraestrutura e seu contexto dentro da arquitetura.

A camada de infraestrutura é composta por um conjunto de classe pertencentes à camada de acesso a dados, responsável por tratar as toda a requisição de acesso a dados provenientes da camada de regra de negocio, feita através do uso de uma interface DAO entre a camada de negócios e a camada de modelo. Dessa forma, as principais funcionalidades implementadas são: inserção, exclusão, alteração e consulta.

Com a criação do DAO, separamos o acesso a elementos de persistência, assim como o controle e a manipulação dos dados que são retornados de um banco de dados.

* 1. Domínio

Onde é implementada a regra de negócio?

Todo sistema é formado por um conjunto de regras de negócio, ou seja, um fluxo lógico que deve ser processado para que tenhamos o resultado desejado. Uma regra de negócio é representada na UML (linguagem de modelagem de sistemas orientada a objetos) através de um caso de uso (modelagem da regra de negócio). Com a análise orientada a objetos a preocupação com a separação real da regra de negócio das demais funcionalidades do sistema é constante e essencial para termos um sistema de baixo acoplamento e de fácil manutenção e extensão.

Por esse motivo foi criado o elemento chamado controle dentro da camada de domínio na arquitetura. Um controlador é responsável pela execução de um ou mais fluxos de execução que são modeladas em um caso de uso, ou seja, podemos dizer que o controlador é em si a implementação da regra de negócio. O mesmo pode ser modularizado, quando existem algumas particularidades dentro da implementação das regras, em classes que chamamos de RN (regras de negócio). O controlador faz uso da infraestutura para obter as informações necessárias para o seu processamento.

Com isso temos para cada caso de uso existente no sistema um controlador responsável por implementá-lo, assim temos um controle transacional muito mais robusto (por caso de uso ou pela interação entre eles), por exemplo, cada método dentro do controlador estará sempre sobre o mesmo contexto transacional.

* 1. Visão

Como é dividida? Quais seus principais elementos?

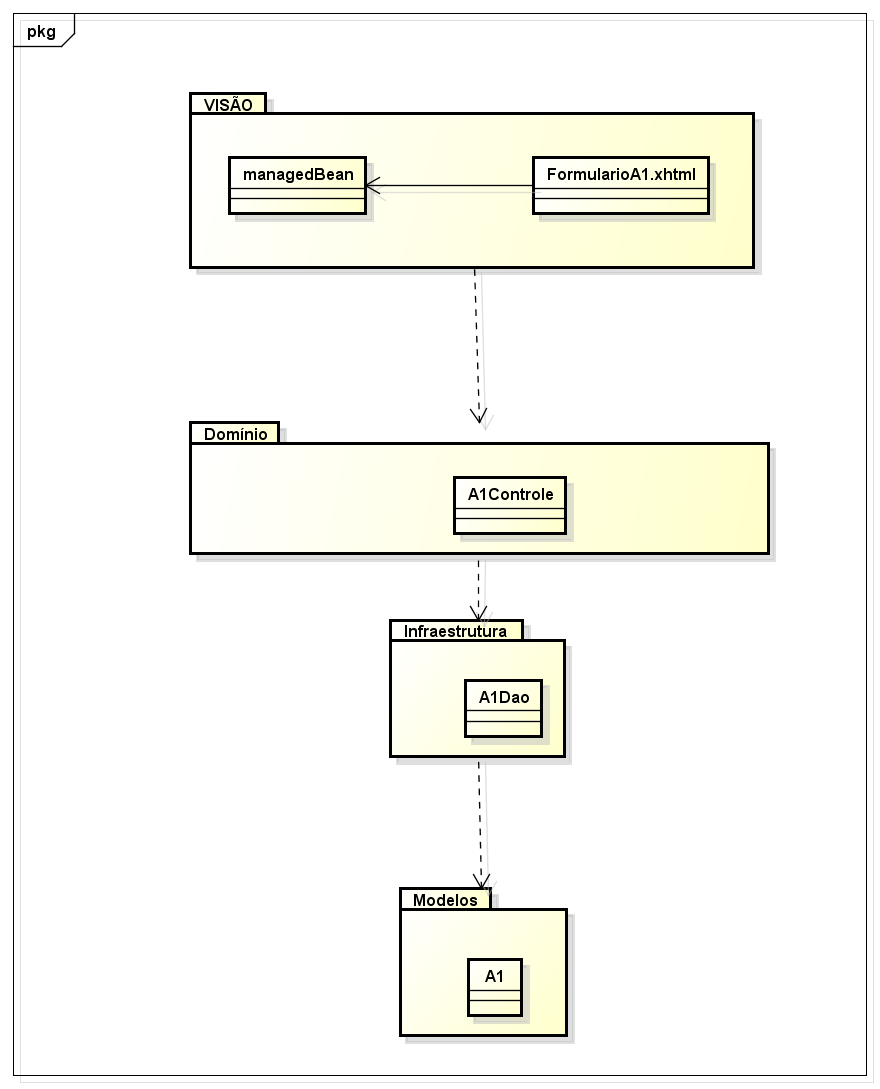
Nós já detalhamos bem a camada de regra de negócio e persistência de dados, agora iremos detalhar a camada de interface com o usuário. Nesse primeiro momento iremos falar sobre interfaces web, como ela foi dividida e o que foi envolvido na criação da mesma.

O GlassFish é um servidor de aplicações Java para web. É oficialmente endossado pela Sun como a Implementação de Referência (RI) para as tecnologias Java Servlet e JavaServer Faces (JSF). O GlassFish é robusto e eficiente o suficiente para ser utilizado mesmo em um ambiente de produção.

Parte da plataforma corporativa Java Enterprise Edition (J2EE ou Java EE) que abrange as tecnologias Servlet e JSP,JSF, incluindo tecnologias de apoio relacionadas como Realms e segurança, JNDI Resources e JDBC DataSources.

A interface web foi desenvolvida utilizando-se JSF.

Desenho GERAL da A rquitetura



# Padrões de Projeto

# Objetivos e Restrições Arquiteturais

* 1. Requisitos básicos
  2. Estratégias de implementação