UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

REENGENHARIA DAS APLICAÇÕES *WEB* DO SISTEMA IRRIGA

TRABALHO DE GRADUAÇÃO

Elias Dorneles da Silveira Junior

Santa Maria, RS, Brasil

2008

REENGENHARIA DAS APLICAÇÕES *WEB* DO SISTEMA IRRIGA

por

Elias Dorneles da Silveira Junior

Trabalho de Graduação apresentado ao Curso de Ciência da Computação
da Universidade Federal de Santa Maria (UFSM, RS), como requisito
parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Prof. Dr. Benhur de Oliveira Stein

Co-orientador: Prof. Msc. Celio Trois (CEFET-SVS)

Trabalho de Graduação Nº 271
Santa Maria, RS, Brasil

2008

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

REENGENHARIA DAS APLICAÇÕES *WEB* DO SISTEMA IRRIGA

elaborado por
Elias Dorneles da Silveira Junior

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Prof. Dr. Benhur de Oliveira Stein
(Presidente/Orientador)

Prof. Msc. Antonio Marcos de Oliveira Candia (UFSM)

Prof. Dr. Raul Ceretta Nunes (UFSM)

Santa Maria, 17 de dezembro de 2008.

AGRADECIMENTOS

Agradeço a meus pais e minha família por sempre me apoiarem, mesmo nas minhas loucuras.

Agradeço aos professores, em especial ao meu orientador Benhur Stein por me aturar, e ao professor Raul Ceretta por sua solicitude em ajudar. Agradeço também à Nelma e à Janice, pela infinita paciência.

Agradeço ao meu chefe, Reimar Carlesso, por seu apoio, e aos rapazes Brantan Chagas e Celio Trois, por serem comigo a melhor equipe de todos os tempos. Voceis ruleiam!

Agradeço ao João Vicente Lima (raquer!) pela bela epígrafe, e ao Valdir Stumm Júnior (ilustre desconhecido) por ter salvo, entre outras coisas, minha vida, meu curso e minha parvalhice com sua lata de psicopata, seu legado e sua teoria do humor, respectivamente. Valeu!

Agradeço aos colegas Marília, Carol, Dewes e Douglas pela amizade e parceria.

Agradeço à Ana Cristina, por tudo.

Finalmente, agradeço ao Machadão, pelo melhor xis e maionese da cidade.

Amém!

“HAU!”
— JOÃO VICENTE FERREIRA LIMA

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

REENGENHARIA DAS APLICAÇÕES *WEB* DO SISTEMA IRRIGA

Autor: Elias Dorneles da Silveira Junior
Orientador: Prof. Dr. Benhur de Oliveira Stein
Co-orientador: Prof. Msc. Celio Trois (CEFET-SVS)
Local e data da defesa: Santa Maria, 17 de dezembro de 2008.

As aplicações Web possuem um ciclo de vida e de desenvolvimento singular na área de engenharia de software. Nesse tipo de aplicação, é favorecido o desenvolvimento rápido e o menor esforço na produção de resultados, pois os requisitos e restrições tendem a ser alterados com maior freqüência, somando-se ao fato das tecnologias nesta área estarem em constante evolução. Esse tipo de desenvolvimento, se não for planejado e desenvolvido de acordo com padrões que permitam compatibilidade e extensibilidade, pode levar a um estado em que o sistema se torna muito difícil de ser mantido e em alguns casos, de ser utilizado. O Sistema Irriga é um conjunto de serviços tecnológicos de manejo e monitoração de culturas agrícolas, cujas aplicações Web foram desenvolvidas por pessoas com formações distintas em diferentes períodos de tempo, o que acarretou uma grande heterogeneidade no código-fonte e na interface das aplicações. Essas características dificultam a manutenção do código existente e a adição de funcionalidades, de tal forma que com o surgimento de novas necessidades no projeto, optou-se por realizar uma reengenharia do sistema atual, com o objetivo de resolver os problemas identificados e possibilitar a adoção de novas tecnologias através de um sistema modular e extensível. O presente trabalho apresenta a proposta da reengenharia dessas aplicações Web, fazendo uso de técnicas de engenharia de software e mantendo o desenvolvimento das aplicações de acordo com os padrões atuais.

Palavras-chave: Aplicações web, reengenharia de sistema, engenharia de software, manejo de irrigação.

ABSTRACT

Graduation Work
Graduate Program in Computer Science
Federal University of Santa Maria

REENGINEERING OF SISTEMA IRRIGA WEB APPLICATIONS
Author: Elias Dorneles da Silveira Junior
Advisor: Prof. Dr. Benhur de Oliveira Stein
Coadvisor: Prof. Msc. Celio Trois (CEFET-SVS)

Web applications have a very particular development and life cycle in software engineering. In this kind of application, there are usually favouring of fast development and minimum effort in producing results, as the requirements and restrictions tend to be changed very often. If this development isn't planned nor developed following some standards to allow compatibility and extensibility, it may leave the system in a state that it's very difficult to maintain, and in some cases, to be actually useful. Sistema Irriga is a set of technological services of irrigation management and monitoring, whose Web applications were developed by people with distinct formations and in different times, which has caused an inherent heterogeneity through their source code and also their interface. These characteristics makes it very difficult to maintain the existing code and to add funcionality to the system in such a way that, with the arise of new requirements for the project, it was opted to accomplish a reengineering of the current system, aiming to solve the identified problems and allow the adoption of new technologies, using a modular and extensible system. This work documents the reengineering proposed of those Web applications, making use of software engineering techniques and bringing the applications development to the current standards.

Keywords: web applications, software system reengineering, software engineering, irrigation management.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
AJAX	Asynchronous Javascript And XML
CSS	Cascading Style Sheet
URI	Uniform Resource Identifier
URL	Uniform Resource Location
MVC	Model View Controller
XML	eXtensible Markup Language
PHP	PHP: Hypertext Preprocessor
SQL	Structured Query Language
CI	CodeIgniter
IP	Internet Protocol

SUMÁRIO

1 INTRODUÇÃO

O Sistema Irriga é um projeto existente desde 1999, que consiste em um conjunto de serviços tecnológicos de manejo e monitoração da irrigação de culturas agrícolas, sendo criado, desenvolvido e mantido por uma equipe de pesquisadores e estudantes da Universidade Federal de Santa Maria, e coordenado pelo professor Reimar Carlesso, do departamento de Engenharia Rural dessa Universidade (SISTEMA IRRIGA, 2000).

Para execução destes serviços, o Sistema Irriga mantém uma base de dados de meteorologia, análises de solos e informações sobre plantas e equipamentos. Sobre esta base, o Sistema Irriga executa simulações baseadas em modelos matemáticos que permitem a previsão de irrigações futuras, com o objetivo de otimizar a utilização de água e de energia, evitando consumos desnecessários (CARLESSO; PETRY; TROIS, 2007).

A interface primária do sistema com os usuários, tanto administrativos como clientes, é feita através de um sistema Web. Essa interface foi desenvolvida à medida que as necessidades do projeto foram surgindo, sem muita preocupação com a definição de um padrão ou com o uso de técnicas adequadas de desenvolvimento de software. Em virtude disto, o sistema hoje apresenta algumas deficiências e falta de padronizações no código, dificultando a adição de funcionalidades novas e a manutenção das já existentes.

Propõe-se neste trabalho a reengenharia das aplicações Web do Sistema Irriga, fazendo uso de técnicas da engenharia de software e de ferramentas de apoio ao desenvolvimento, realizando um planejamento com o objetivo de facilitar as atividades de manutenção e adição de funcionalidades, visando também a otimização da utilização dos recursos e tecnologias disponíveis.

Este texto descreve o desenvolvimento desse trabalho até o presente momento (novembro de 2008). O capítulo 2 apresenta conceitos e tecnologias que fazem parte do Sistema Irriga atual, descrevendo seu funcionamento e suas deficiências, e também conceitos

e tecnologias que devem ser utilizados na solução desses problemas na reengenharia proposta nesse trabalho. O capítulo 3 contém a proposta de uma solução, descrevendo seus objetivos e a abordagem que deve ser utilizada para eles serem alcançados. A seguir, o capítulo 4 apresenta os resultados e detalhes da implementação da solução, descrevendo os problemas enfrentados e as soluções utilizadas. Finalmente, o capítulo 5 conclui o trabalho e lista possíveis projetos futuros relacionados.

2 CONCEITOS E TECNOLOGIAS

Este capítulo apresenta conceitos e tecnologias presentes nas aplicações Web do Sistema Irriga atual, com descrições dos problemas existentes os quais este trabalho se dedica a resolver. A seguir, são apresentados conceitos e tecnologias que são utilizados na implementação da reengenharia proposta nesse trabalho.

2.1 Sistemas de manejo de irrigação

A determinação do momento oportuno para efetuar a aplicação de água e da quantidade a ser aplicada é chamada de manejo de irrigação, sendo uma técnica de grande importância para a produção de culturas agrícolas específicas, além de possibilitar um menor consumo de água e energia. Para realizar essa determinação, é necessário levar em consideração fatores como o clima, o tipo de solo, a planta sendo cultivada e o equipamento usado na irrigação, destacando-se o clima como fator determinante da necessidade de água das plantas. Assim, o manejo de irrigação de uma determinada cultura pode ser executado fazendo uso de medidas de radiação solar, temperatura do ar, umidade relativa do ar e velocidade do vento, que influenciam diretamente nas perdas de água das plantas para a atmosfera e, por conseguinte, na necessidade de água das plantas (CARLESSO; PETRY; TROIS, 2007; SISTEMA IRRIGA, 2000).

2.1.1 Sistema Irriga

O Sistema Irriga teve por objetivo principal desenvolver um sistema de manejo de irrigação que permitisse de forma rápida e fácil sua aplicação no campo. Para isso, foi criado um mecanismo de coleta e transmissão de dados meteorológicos a partir de estações fixas nos locais que se tem interesse, como ilustrado na figura 2.1.

Os dados meteorológicos enviados aos servidores são armazenados em um banco de

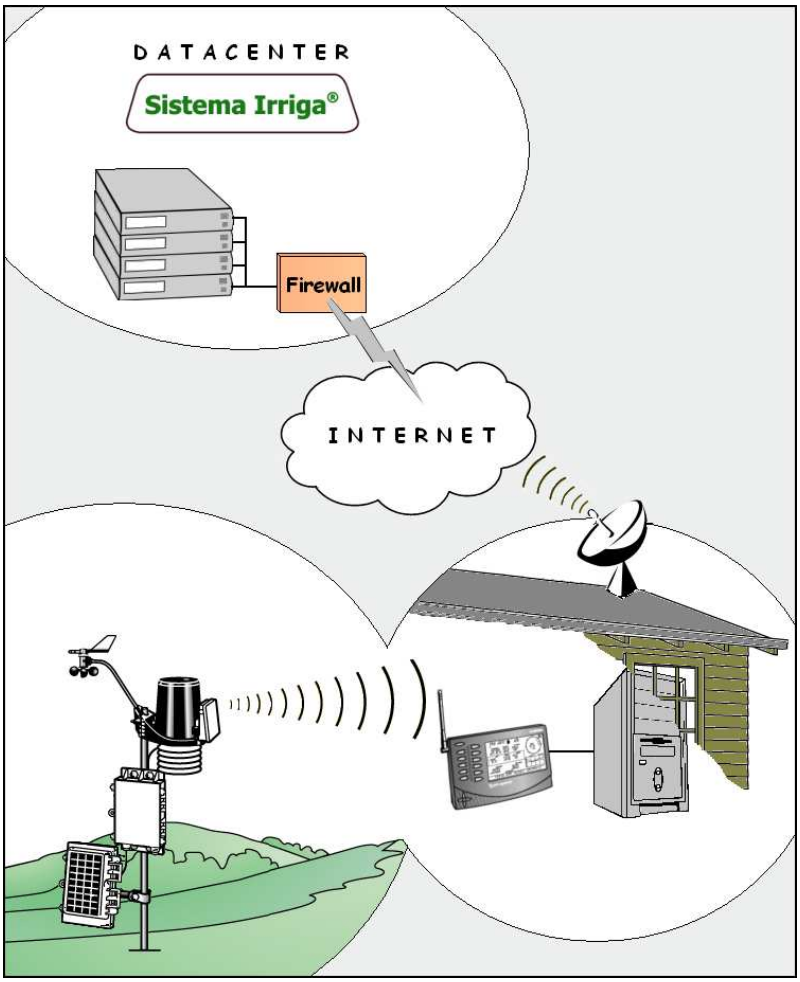

Figura 2.1: Coleta e transmissão de dados meteorológicos da PCD (CARLESSO; PETRY; TROIS, 2007)

dados MySQL, e depois de validados são utilizados para efetuar o cálculo da determinação de quando e quanto se deve irrigar, para as áreas ativas configuradas. Esse cálculo é realizado por um *script* PHP, que recupera do banco MySQL esses dados meteorológicos, juntamente com os resultados de análises físicas de solo, as configurações dos sistemas de irrigação e dados das precipitações pluviais e, com base nesses dados, executa modelos matemáticos que geram recomendações de irrigação para o dia corrente e previsões de irrigação para os dias seguintes. Esses resultados são então, disponibilizados para os usuários em uma interface Web, ilustrada na figura 2.2.

As aplicações Web do Sistema Irriga consistem nas interfaces de cadastros e configurações dessas informações, e de acesso aos dados dos resultados de recomendação e previsão para os usuários irrigantes através de um portal disponível pela rede mundial através dos endereços http://irriga.proj.ufsm.br e http://www.sistemairriga.com.br.

Essas aplicações Web foram desenvolvidas utilizando a linguagem PHP no lado ser-

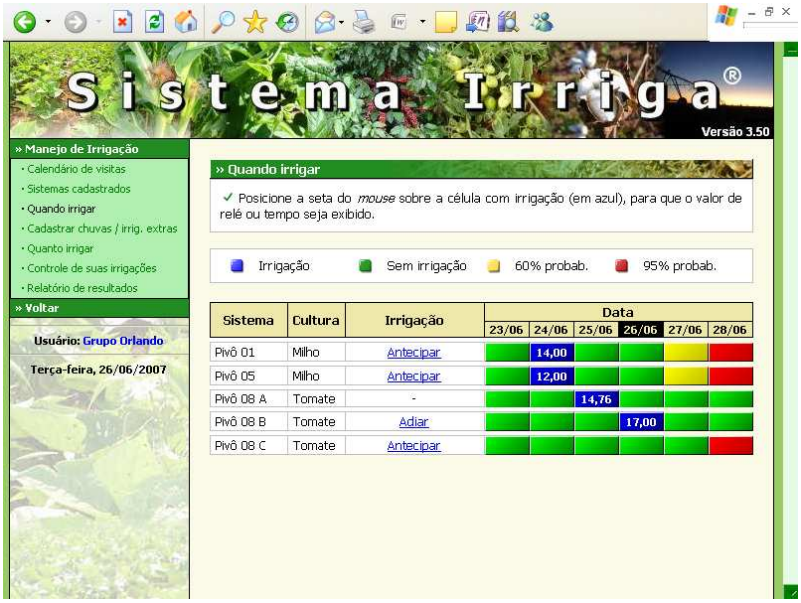

Figura 2.2: Captura da interface do Sistema Irriga disponível aos usuários irrigantes, indicando quando e quanto irrigar para cada sistema de irrigação cadastrado (CARLESSO; PETRY; TROIS, 2007)

vidor e JavaScript no lado cliente, sendo servidas através de um servidor HTTP Apache em uma máquina rodando uma distribuição do GNU/Linux. Os elementos envolvidos no funcionamento dessas aplicações está representado na figura 2.3.

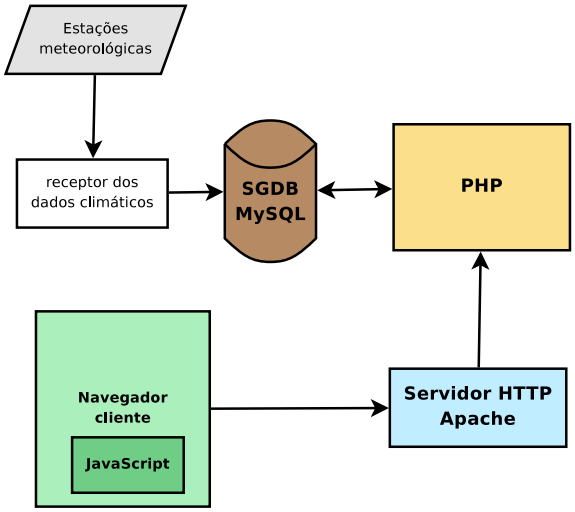

Figura 2.3: Esquema de funcionamento das aplicações Web do Sistema Irriga

O desenvolvimento destas aplicações, porém, não foi planejado de maneira adequada, de maneira que hoje o sistema apresenta uma série de deficiências que dificultam a sua manutenção e o desenvolvimento de melhorias. As seções seguintes deste capítulo apresentam algumas deficiências as quais a reengenharia proposta nesse trabalho busca resolver, e os elementos que deverão ser utilizados nas soluções.

2.2 Problemas de interface e projeto

A seção do portal do Sistema Irriga, uma aplicação Web disponível apenas para os operadores internos (equipe do Sistema Irriga e alguns representantes credenciados), é onde estão todas as interfaces de cadastro e configurações de estações, sistemas de irrigação, áreas contratadas, análises físicas de solo, precipitações pluviais, ajustes e correções dos dados para o cálculo, usuários e permissões de acesso, e também visualizações de relatórios sobre o estado das estações, médias dos dados meteorológicos e resultados do cálculo.

Devido ao fato de essas funcionalidades terem sido desenvolvidas por pessoas com diferentes formações, em períodos de tempo intercalados, com pouca ou nenhuma interação, o código da aplicação e a interface com o usuário são bastante heterogêneos. Há trechos em que as interfaces são desenhadas de forma elegante, porém suas implementações foram baseadas em *templates* PHP, com um código monolítico utilizando apenas estruturas de repetição e seleção, com ausência de abstrações através de funções ou de orientação a objetos, e excesso de redundância de código. Por outro lado, há módulos cujo código faz uso de abstração, separando a lógica em funções e, em alguns casos, utilizando classes e orientação a objetos, porém suas interfaces com o usuário sofrem de problemas de usabilidade, com JavaScripts quebrados, botões posicionados de forma desordenada, e em lugares diferentes em cada tela.

No Sistema Irriga atual, todas as requisições para a aplicação Web são feitas utilizando do método HTTP POST, devido a uma decisão de projeto tomada na fase inicial de desenvolvimento, na tentativa de contornar o fato do protocolo HTTP ser *stateless* (não mantém informações sobre os clientes durante as requisições) (FIELDING et al., 1999). Mais tarde, essa decisão se revelou problemática, com a manifestação de uma falha de segurança que permitia acesso não autorizado ao sistema, problema que até hoje não foi completamente contornado, ainda persistindo em alguns casos específicos. Além disso, também conseqüência dessa decisão, os recursos do sistema são impossíveis de serem endereçados com um URI, o que é inconveniente pois impede o usuário de adicionar um recurso aos favoritos ou compartilhá-lo com outro usuário (JACOBS; W3C, 2004).

Essas características afetam profundamente o sistema, dificultando a adição de novas funcionalidades devido às limitações impostas pelas decisões problemáticas e a alta densidade do código.

2.3 JavaScript

JavaScript é uma linguagem de *scripting* amplamente utilizada em aplicações Web para estender as suas funcionalidades rodando no navegador no lado cliente. Esta linguagem possui várias conveniências na criação de aplicações Web, permitindo às aplicações oferecer um nível de experiência ao usuário próximo ao de aplicações sendo executadas localmente (MOZILLA FOUNDATION, 2008).

Apesar da disponibilidade de JavaScript na maioria dos *browsers* modernos, a dependência dessa tecnologia estar disponível pode consistir em um problema de acessibilidade para uma aplicação. Isso ocorre porque o funcionamento do sistema fica desabilitado no caso de ausência de um interpretador disponível (e.g. *browsers* de celulares, robôs de pesquisa, *browsers* destinados a pessoas com deficiência visual, *browsers* em modo texto) ou caso a interpretação de JavaScript esteja desabilitada ou permitida somente para *sites* em domínios conhecidos (comum prática de segurança para evitar execução local de código malicioso externo). Além disso, *browsers* modernos com implementações completas da linguagem não são completamente compatíveis, havendo alguns recursos que não foram padronizados e alguns *browsers* que não seguem as especificações (KORPELA, 2005).

As aplicações Web do Sistema Irriga atual são dependentes da disponibilidade de JavaScript no navegador cliente, sendo impossível serem utilizadas sem a presença de um interpretador. No sistema atual, é utilizado JavaScript para geração de campos em formulários no tempo de carregamento da página pelo browser, submissão de formulários a partir de eventos do navegador e, em alguns módulos, para a validação dos dados digitados nos formulários, o que consiste em uma potencial brecha de segurança, para o caso de serem submetidos dados com código malicioso com a interpretação JavaScript desabilitada. Além disso, a maior parte do código JavaScript atual foi desenvolvido primariamente para o navegador Internet Explorer, sofrendo de vários problemas de compatibilidade com outros *browsers*.

A figura 2.4 ilustra alguns desses problemas: todos os botões da interface mostrada na figura são elementos HTML com eventos JavaScript associados a métodos que ocasionarão a submissão de um formulário contendo uma série de variáveis, necessárias para a aplicação manter o estado da sessão.

O uso da abordagem AJAX para enriquecimento da interface dessas aplicações está atualmente impedido devido a maneira que os códigos JavaScript estão sendo acionados,

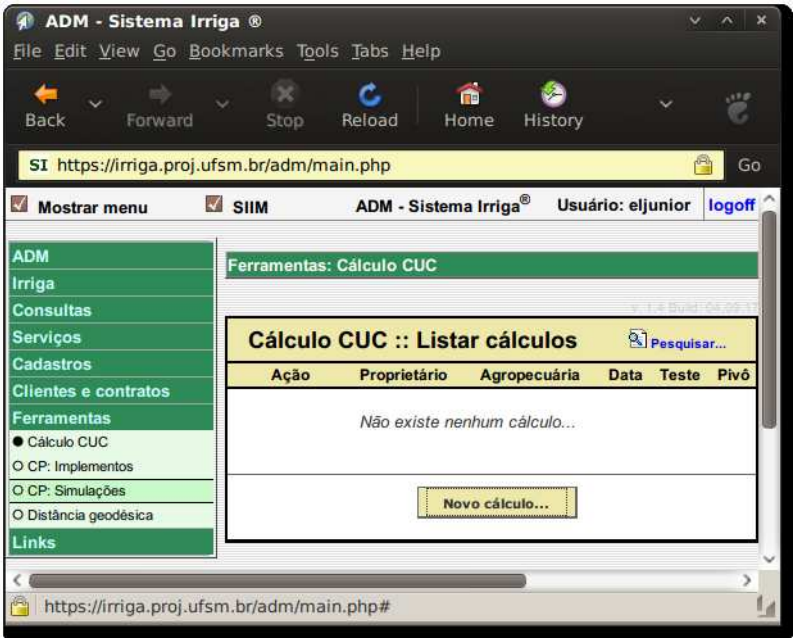

Figura 2.4: Captura de tela do sistema atual: botões com JavaScript para submeter formulário

durante o carregamento da página, inseridos em um bloco HTML *script*. Isso impede a execução desses scripts porque os navegadores não permitem a execução de código JavaScript injetado através da propriedade *innerHTML*, utilizada para atualizar a página após a execução de uma requisição *XMLHttpRequest* (Mozilla Developer Center, 2008; MICROSOFT CORPORATION, 2008).

2.3.1 jQuery

A jQuery é uma biblioteca JavaScript que permite o uso desta linguagem de maneira não obstrutiva e totalmente em separado do código de marcação. Esta biblioteca inclui diversas funcionalidades que agilizam a criação de código para a manipulação dos elementos de uma página, através de uma extensa biblioteca de funções, mantendo a compatibilidade com os navegadores mais utilizados (RESIG; jQuery Team, 2008a).

Assim, o uso dessa biblioteca para as funcionalidades JavaScript de forma não obstrutiva permite que seja efetuada uma degradação graciosa para os casos de indisponibilidade de JavaScript no cliente, garantindo acessibilidade à aplicação em todos os *browsers* (WILLISON, 2008).

2.4 Internacionalização

Atualmente, a atuação do Sistema Irriga é em maior escala no Brasil. Porém, estão em andamento projetos de implantação do Sistema em outros países, havendo já algumas áreas implantadas no Uruguai há alguns anos. A necessidade de internacionalização, portanto, não é novidade para o sistema. Contudo, o problema de localização no sistema atual foi resolvido diretamente no código, com um vetor funcionando como dicionário, contendo todas as *strings* e suas respectivas traduções. Essa solução caracterizou-se como de manutenção trabalhosa, e foi aplicada apenas para a seção do *site* acessada pelos produtores. A seção do *site* restrita à equipe do Sistema Irriga e aos representantes dos produtores está totalmente em português, obrigando os representantes oriundos de outros países a usarem o sistema nesse idioma.

2.4.1 GNU gettext

O pacote de utilitários GNU Gettext é um conjunto de ferramentas que provê um *framework* com o qual se podem construir programas com suporte multi-lingual. Ele consiste em uma série de convenções sobre como os programas devem ser escritos para suportar catálogos de mensagens, bibliotecas permitindo a análise dos catálogos e recuperação das mensagens traduzidas, e programas para geração e manipulação dos catálogos de mensagens (FOUNDATION, 2008).

Existe um *binding* da GNU gettext disponível para a linguagem PHP, com uma interface simplificada das funções de manipulação de catálogos, a PHP Gettext. O uso das funções da PHP-gettext permite a simplificação do processo de internacionalização de uma aplicação Web, tornando a tarefa de localização trivial com o uso de ferramentas externas como o Poedit para tradução dos catálogos de mensagens (PHP Documentation Group, 1997; POEDIT, 2008).

2.5 Model-View-Controller

A abordagem *Model-View-Controller* (MVC) para construção de interfaces com o usuário consiste na utilização de três tipos de objetos: o objeto de aplicação (Modelo), a apresentação na tela (Visão) e as definições de como a interface do usuário reage às entradas (Controlador). A MVC faz uso da separação desses elementos para aumentar a flexibilidade e a reutilização de código, permitindo a ligação de múltiplas visões a um

modelo para fornecer diferentes apresentações (GAMMA et al., 2000).

2.5.1 CodeIgniter

O CodeIgniter (CI) é um *framework* de código aberto voltado para a construção de aplicações Web na linguagem PHP que faz uso da abordagem *Model-View-Controller* de uma maneira particular, a fim de permitir o desenvolvimento rápido de aplicações mantendo mínimo o código-fonte dos *scripts*. Numa aplicação típica com essa abordagem, as classes dos modelos contêm funcionalidades de recuperação, adição e atualização das informações no banco de dados, as visões são *templates* PHP que renderizam conteúdo (páginas Web, dados em XML, etc.), e as classes dos controladores contém a lógica de processamento das requisições HTTP e geração das páginas (ELLISLAB, 2008).

Acompanha o *framework* uma coleção de bibliotecas e *plugins* para facilitar tarefas comuns na criação de aplicações Web, que permitem ampla extensibilidade e mesmo a substituição completa de suas funcionalidades, através de um mecanismo para facilitar o carregamento e utilização dessas bibliotecas. Utilizando essas ferramentas, o *framework* incorpora funcionalidades de roteamento de URLs, cacheamento de páginas, *benchmarking* e depuração, validação e filtragem dos dados de entrada, entre outras, de forma bastante flexível, com a facilidade de extensão destas funcionalidades e de adição de outras (ELLISLAB, 2008).

A figura 2.5 ilustra a interação dos elementos que compõem uma aplicação com o *framework* CodeIgniter.

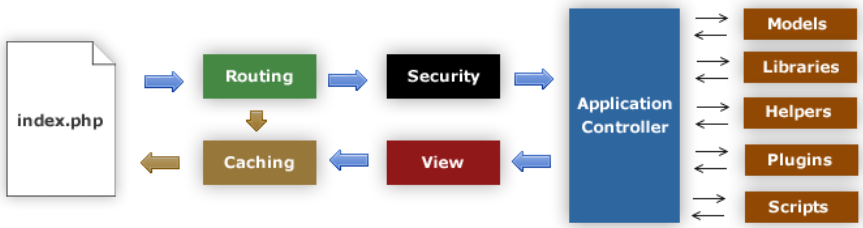

Figura 2.5: Diagrama de fluxo de uma aplicação utilizando o *framework* CodeIgniter (ELLISLAB, 2008)

Dessa forma, a utilização do CodeIgniter para construção de uma aplicação Web permite a redução da quantidade de código e a reutilização de código, através de um conjunto de bibliotecas e uma estrutura de funcionamento flexível, reduzindo, assim, o esforço necessário para o desenvolvimento e manutenção das aplicações.

3 SOLUÇÃO PROPOSTA

Este capítulo apresenta uma proposta para solucionar os problemas das aplicações Web do Sistema Irriga através de uma reengenharia do sistema atual. Primeiramente serão listados os objetivos do trabalho, e a seguir é descrito em maiores detalhes como eles foram implementados.

3.1 Objetivos do Trabalho

De forma resumida, os objetivos específicos podem ser listados como:

- projetar a interface Web para funcionar independente de JavaScript;
- adicionar suporte a internacionalização a todas as páginas Web;
- fazer uso de orientação a objetos e padrões de projeto para maximizar a reutilização de código;
- utilizar consistentemente os métodos de requisição HTTP GET e POST de forma a permitir endereçabilidade de recursos úteis;
- criar e documentar padrões de codificação para a base de código do sistema;
- criar um protótipo funcional do sistema;

Estes objetivos são melhor explicados nas seções a seguir.

3.2 Independência de JavaScript

Um dos objetivos na reengenharia do sistema é tornar o sistema independente de JavaScript, a fim de mantê-lo acessível a partir de qualquer *browser*. As funcionalidades do JavaScript devem ser adicionadas de forma não obstrutiva, isto é, sem se tornarem

essenciais para o funcionamento da aplicação. Para isto, o código HTML deve ser feito marcando os elementos do documento a fim de permitir a manipulação dos elementos via JavaScript. Para isso, se utiliza métodos de seleção de elementos por identificador (atributo HTML *id*), por classe (atributo HTML *class*), e por posição na hierarquia do documento (por exemplo, o menu pode ser um elemento lista não ordenada *ul* cujos filhos (items de lista *li* sejam *links* para outras páginas).

Manter o sistema independente de JavaScript tornará o desenvolvimento do sistema de forma mais organizada, por causa da separação de interesses, pois o código JavaScript será totalmente separado do código HTML e do código PHP. Além disso, um sistema funcional sem JavaScript possibilita o funcionamento em *browsers* minimalistas como os presentes em celulares e outros dispositivos móveis.

3.2.1 Biblioteca jQuery

A jQuery é uma biblioteca JavaScript que provê uma interface de programação para especificar elementos de um documento HTML através de uma função de consulta (*query*) que utiliza uma linguagem baseada em CSS e XPath, e para manipular os elementos especificados, através de uma série de métodos encapsulados no objeto resultante da consulta. (RESIG; jQuery Team, 2008a,b).

Essa biblioteca foi escolhida para atingir a independência de JavaScript, devendo ser utilizada para adicionar funcionalidades no lado cliente de forma não obstrutiva, isto é, sem obstruir o uso da aplicação em clientes sem disponibilidade de JavaScript, pois seu uso permitirá manter a compatibilidade entre os *browsers* com menos esforço e em uma base de código reduzida. (WILLISON, 2008)

3.3 Suporte a internacionalização

O suporte a internacionalização na reengenharia do sistema deve ser implementado utilizando catálogos de mensagens no formato da GNU gettext e as funções da PHP-gettext para recuperar as mensagens no idioma adequado. O código-fonte deverá usar a codificação de caracteres Unicode UTF-8, e as mensagens no código fonte deverão ser mantidas no idioma inglês. A tradução das mensagens será feita mais tarde, por uma pessoa com capacidade para traduzir para os idiomas desejados, utilizando uma ferramenta para edição de catálogos.

3.4 Convenções e reuso de código

Na construção das aplicações deve ser utilizado o *framework* CodeIgniter, adotando sua abordagem para o *Model-View-Controller*, separando o código das estruturas de dados do código de apresentação, utilizando seu mecanismo para criação de bibliotecas e *plugins* para as funcionalidades da aplicação e estendendo as existentes conforme a necessidade, e seguindo o estilo de codificação sugerido na documentação do *framework*. Dessa forma, será possível obter maior padronização na codificação, dada a estrutura definida com o *framework*, e maior reuso de código, facilitando as tarefas de desenvolvimento e manutenção do sistema. Este *framework* foi escolhido porque, além das características mencionadas, ele permite de forma fácil a reutilização do código existente.

3.4.1 Uso de HTTP GET e POST

Os métodos de requisição HTTP GET e POST devem ser utilizados de maneira que seja possível o endereçamento de recursos úteis via URIs utilizando o método GET, e que operações que façam alterações no estado do sistema (inclusão ou atualização de informações) utilizem o método POST, seguindo as recomendações em documentação do W3C. (JACOBS; W3C, 2004) Seguir essas recomendações permitirá as vantagens da adressabilidade via URIs, e manter dados potencialmente sensíveis fora da URI. Além disso, após o processamento de cada submissão via POST, deve ser feito um redirecionamento HTTP. Isto é necessário para evitar o reenvio de dados e a conseqüente repetição desnecessária da operação, quando o usuário requisitar a atualização (*refresh*) da página.

A implementação dessas características da aplicação deve ser feita nos componentes controladores (*Controllers*) da aplicação, e amparado por um mecanismo para manter o estado da sessão do usuário.

3.5 Criação de protótipo

Finalmente, deve ser implementado um protótipo funcional do sistema com as características citadas anteriormente, com a finalidade de testar a efetividade da reengenharia proposta, e ser utilizado como base para a construção das aplicações. O projeto do protótipo está subdividido nos seguintes módulos:

- Módulo de *Administração do Sistema*: deve conter os cadastros de recursos do sistema, controle dos usuários e permissões de acesso, e *logs* de acesso.

- Módulo de *Cadastro de Estações*: deve conter os cadastros de estações, grupos de estações e configurações dos dados meteorológicos.
- Módulo de *Cadastro de Áreas*: deve conter os cadastros de sistemas de irrigação, pivôs, subáreas de pivôs, precipitações, correções e ajustes dos resultados.
- Módulo de *Cálculo do Manejo*: deve conter as funções para calcular a recomendação de irrigação para o dia e a previsão para os dias seguintes, utilizando os mesmos parâmetros agronômicos e modelos matemáticos do sistema atual.

Este protótipo servirá como modelo de produção, de forma a permitir a construção das aplicações a partir da sua base de código, e eventualmente, permitir a migração para estas aplicações, substituindo o sistema atual.

4 RESULTADOS

Este capítulo apresenta a implementação da solução proposta, descrevendo o estado atual desta, apontando os itens que não foram possíveis ser completados e por quê.

4.1 Independência de JavaScript

Para obter a independência de JavaScript, fazendo uso dessa tecnologia como uma camada de melhoramentos sobre as aplicações, seguiu-se o procedimento descrito abaixo:

1. criação da aplicação utilizando HTML semântico, isto é, não usar HTML para formatação;
2. adição de estilização visual CSS em arquivo externo; e
3. adição de código JavaScript em arquivo externo aplicando melhorias no comportamento do *site*.

O uso de HTML semântico, marcando significativamente os elementos e a estrutura do documento, é de muita importância para a execução do último passo pois permite a utilização de mecanismos de seleção de objetos do documento HTML para manipulação via JavaScript. A biblioteca jQuery fornece mecanismos de seleção de objetos utilizando os mesmos seletores das folhas de estilos CSS, adicionados de outros seletores complementares (RESIG; jQuery Team, 2008b). Esses seletores tornam possível a configuração de *handlers* de eventos para elementos do documento diretamente no código dos *scripts* externos, permitindo dessa forma o uso não obstrutivo desses recursos.

O protótipo do sistema cuja implementação está em andamento é independente de JavaScript, sendo todas as funcionalidades acessíveis em qualquer navegador, e contendo uma camada de JavaScript com melhorias na interface para *browsers* com um interpretador disponível. Dessa forma, se não houver JavaScript disponível o sistema funciona em

modo degradado, caso contrário, o sistema faz uso de JavaScript para melhorar aspectos de usabilidade e acessibilidade da navegação.

As vantagens dessa abordagem podem ser demonstradas analisando o caso dos menus em abas do sistema, no sistema atual e no protótipo desenvolvido. No sistema atual, as interfaces com abas foram codificadas em tabelas HTML, consistindo de células com imagens e texto intercaladas que, quando clicadas, disparam um evento JavaScript que causa a execução de um método que seta um campo do formulário com a identificação da aba clicada e submete o formulário, enviando os dados. No protótipo criado, as abas são listas HTML cujos elementos são *links*, estilizadas com CSS para terem a aparência em abas desejada. Dessa forma, os menus em abas funcionam em qualquer navegador, independente de ter JavaScript disponível ou habilitado. Além disso, como a marcação semântica é simplificada (listas de *links* em lugar de tabelas com células para texto e imagens), a manipulação do menu também é simplificada: se pode selecionar os elementos do menu, e aplicar operações que permitem alterá-los, movê-los, escondê-los, aplicar efeitos visuais e assim por diante, com apenas uma linha de código jQuery.

4.2 Internacionalização com as ferramentas gettext

O *framework* CodeIgniter, determinado para utilização na implementação deste trabalho, possui uma biblioteca própria (*Language Class*) para suporte a internacionalização. Esta baseia-se na criação de arquivos de dicionários em código na linguagem PHP contendo as mensagens traduzidas, que são carregados seletivamente em tempo de execução. Apesar de funcional, essa abordagem sofre dos mesmos problemas que a abordagem utilizada nas aplicações Web do Sistema Irriga atual, sendo ambas muito semelhantes na forma como utilizam os recursos da própria linguagem PHP para efetuar a tradução, e os *arrays* do PHP para a representação dos dicionários.

Por essa razão, foi implementada uma classe que utiliza as funções da biblioteca PHP-gettext, efetuando a tradução das mensagens em tempo de execução, mantendo a mesma interface da *Language Class* do CodeIgniter. A geração dos catálogos de mensagens e a tradução destas é feita através do programa Poedit, que efetua uma varredura pela base de código existente, atualizando as mensagens dos catálogos, e fornece uma interface para agilizar a tarefa de localização. Dessa forma, o processo de internacionalização das aplicações foi simplificado, restando agora apenas a tarefa de manter as mensagens dos

catálogos atualizadas.

4.3 Implementação utilizando o CodeIgniter e MVC

O *framework* CodeIgniter (CI) está sendo utilizado na implementação desse trabalho, pois oferece diversas funcionalidades comuns em uma aplicação Web (Roteamento de URIs, Mapeador Objeto-Relacional para acesso ao banco de dados, Validação e Filtragem de dados de entrada, assistentes para geração de HTML, Caching), e permite a manutenção de um padrão de codificação, utilizando a abordagem *Model-View-Controller*. A implementação desse trabalho utilizando o CI consiste na implementação de Modelos (classes PHP contendo a manipulação das estruturas de dados e a interação com o banco de dados), Visões (templates PHP que geram páginas Web) e Controladores (classes especiais cujos métodos representam recursos do sistema expostas através de URIs pelo mecanismo de roteamento do CI), e de bibliotecas e *plugins* para alguns recursos adicionais.

4.3.1 Segurança: Sessão, Autenticação e Autorização

Um dos primeiros aspectos a ser implementado foi a autenticação dos usuários do sistema, através da criação de uma biblioteca (*Auth*). Essa biblioteca consiste numa classe que encapsula métodos de autenticação utilizando dados providos pelo usuário ou de um usuário autenticado na sessão. Também, por questões de conveniência, o método de configuração do idioma da sessão foi adicionado nessa mesma classe. No decorrer dessa implementação, desejou-se alterar o modo de funcionamento do mecanismo de sessões do CI, por razões de escalabilidade e segurança das informações da sessão. O mecanismo de sessão do CI grava os dados da sessão (identificador, endereço IP do cliente, identificador do cliente HTTP, timestamp da última atividade e dados adicionados pela aplicação) serializados em um cookie na máquina do cliente. Essa abordagem é inconveniente por não garantir muita segurança, visto que os dados da sessão ficam no cliente, não podendo serem tratados como confiáveis. O *framework* contorna esse problema efetuando a criptografia dos dados no cookie e a verificação das informações de identificação da sessão (identificador, IP do cliente, *useragent* e timestamp) presentes no cookie com as mesmas informações armazenadas em uma tabela no banco de dados, que é atualizada em sincronia com o cookie, assegurando dessa forma a confidencialidade e integridade dos

dados da sessão. Isso acarreta problemas com os dados da sessão adicionados pela aplicação porque o tamanho dos dados de um cookie é limitado pelo *browser* (4 kilobytes é um limite comum), e como os dados criptografados ocupam mais espaço, esse limite é atingido sem muito esforço. Para resolver esses problemas, foi feita uma extensão da biblioteca de sessão do CodeIgniter que guarda os dados da sessão na tabela do banco de dados, utilizando uma coluna para guardar os dados adicionados pela aplicação serializados, e guarda os dados de identificação da sessão no cookie, criptografados. Dessa forma, elimina-se os problemas de tamanho (os dados de identificação da sessão guardados no cookie dificilmente ocupam mais que 1 kilobyte) e de confidencialidade (dados da sessão ficam no servidor) e integridade (apenas os dados de identificação da sessão ficam no cliente, e utilizados para recuperação e verificação dos dados da sessão).

Tendo tratado esses aspectos, fundamentais para a segurança do sistema, a implementação de autorização reduziu-se a criação de um *plugin* contendo um método que verifica se o usuário autenticado tem permissão para acessar o conteúdo do serviço sendo acessado, toma as medidas necessárias (autoriza acesso ou redireciona), e registra o acesso. Dessa forma, foram implementados os aspectos de segurança no sistema, sem comprometer a flexibilidade e escalabilidade das aplicações.

4.3.2 Cadastros, *Formsmart* e gerador de código

Após serem implementados os aspectos de segurança, foi dado início às implementações de cadastros de usuários, e dos recursos do sistema. Para auxiliar essas implementações, foi criada uma biblioteca para criação de formulários HTML, apelidada de *Formsmart*, que encapsula métodos para população dos formulários e de mensagens de erros de validação, renderização dos elementos dos formulários adequadamente populados, rotulados e com as mensagens de erro, se necessário. A *Formsmart* foi desenvolvida favorecendo convenção à configuração, porém mantendo flexibilidade. Ela pode ser usada chamando os métodos específicos para a renderização de campos, ou configurando-se um *array* contendo os dados de cada elemento do formulário (rótulos, identificador no POST, regras de validação, valor *default*, *callback* utilizado para renderização do campo, entre outras opções), deixando a biblioteca renderizá-lo de acordo. A figura 4.1 contém um diagrama que ilustra o funcionamento da *Formsmart* em conjunto com os outros elementos da implementação de um cadastro.

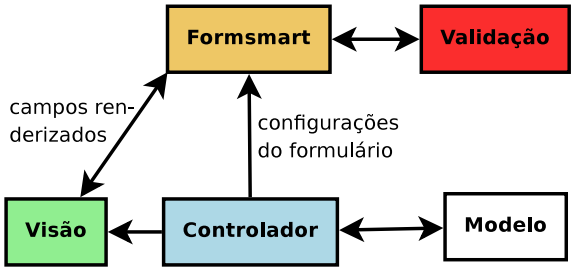

Figura 4.1: Diagrama de interação dos objetos em um cadastro utilizando a biblioteca *Formsmart*

Fazendo uso dessas ferramentas, a implementação dos cadastros tornou-se uma tarefa mais simplificada, sendo reduzida à criação da tabela no banco de dados e à codificação de 4 arquivos: um controlador contendo métodos para listagem, adição, edição e exclusão dos itens do cadastro, e um método com a configuração para a *Formsmart*; um modelo contendo os métodos para recuperação, inserção, atualização e exclusão dos itens; e duas visões (uma para a listagem e outra para adição e edição dos itens). Logo após a construção dos primeiros cadastros, como se tratava de uma tarefa bastante repetitiva, foi decidido criar um sistema para automatização dessa tarefa.

Para atingir esse objetivo, criou-se um *script* que reconhece uma linguagem simples de descrição de tabelas e seus relacionamentos, e gera um código adequado SQL para a criação da tabela no banco e um código PHP necessário para um cadastro funcional (o controlador, o modelo e as duas visões), que pode ser então customizado para as necessidades específicas do cadastro. O código gerado para o controlador contém configurações dos rótulos dos campos, regras de validação, e *callbacks* a serem utilizadas para a renderização do elemento no formulário, de acordo com padrões estabelecidos para o modelo de dados. Alguns nomes de campos têm significados especiais, para registros que mantêm registro de operações, por exemplo, para identificação de usuário efetuando operação, data e hora que determinada ação foi realizada, entre outros. Freqüentemente, a customização de um cadastro com código gerado por esse gerador de código se resume a alteração dos rótulos e das regras de validação no controlador.

Dessa forma, a criação de um cadastro fica resumida à codificação da descrição da tabela na linguagem de descrição criada, e a customização do código gerado, após a geração de código ter sido completada.

4.3.3 Exemplo: captura de tela

A figura 4.2 apresenta uma captura de tela do protótipo, que ilustrar alguns dos problemas que foram abordados na construção deste. Na interface da figura, o sistema está utilizando o idioma Português brasileiro, cuja tradução é obtida de um arquivo contendo o catálogo das mensagens, através das funções da biblioteca PHP-gettext. Os itens do menu (dispostos em abas para melhor aproveitamento da tela), e os botões são *links* HTML comuns, estilizados com CSS, que apontam para os endereços dos recursos a que dizem respeito.

A interface mostrada é a de um cadastro de constantes a serem utilizadas no cálculo para determinadas culturas, que foi gerado com o gerador de código de cadastros. Esse é a interface de listagem do cadastro, e ainda não foi customizada, por isso apresenta alguns defeitos (os rótulos estão iguais aos nomes dos campos no banco de dados, e as chaves estrangeiras estão mostrando o número inteiro em vez de uma string representando o valor). A *Formsmart*, nesse caso, está sendo utilizada apenas para listar os dados.

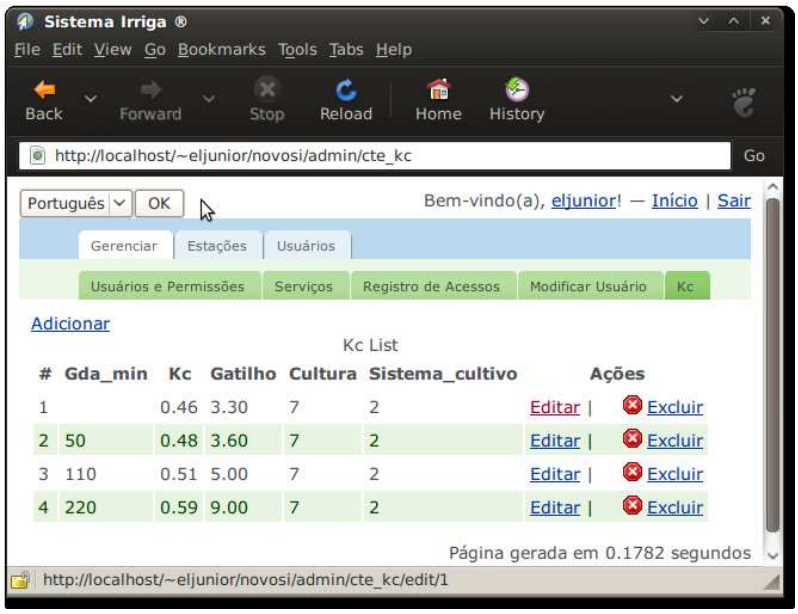

Figura 4.2: Captura de tela do protótipo: endereçabilidade via URIs e JavaScript não obstrusivo

4.4 Situação do Protótipo

A implementação do protótipo, com as características apresentadas, está funcional, com a exceção da previsão de irrigação no módulo do cálculo. O cálculo da previsão é particularmente complexo, pois as regras foram modificadas diversas vezes, ao longo do

tempo, e portanto exigirá mais tempo para ser produzir um código de qualidade.

A seguir, segue uma breve descrição com mais detalhes do estado de cada um dos módulos que foram especificados no capítulo anterior.

Módulo de *Administração do Sistema*. Atualmente, sua implementação está funcional, mas necessita de atualizações na configuração dos recursos, que foram postergadas por necessitar de decisões que podem ser tomadas mais tarde, com melhor aproveitamento.

Módulo de *Cadastros de Estações*. Atualmente, sua implementação está completa.

Módulo de *Cadastros de Áreas*. Atualmente, sua implementação está completa, mas necessita de melhoras na usabilidade.

Módulo de *Cálculo do Manejo*. Atualmente, sua implementação está semi-completa. O cálculo de irrigação para o dia está pronto, e o cálculo de previsão de irrigação está incompleto, devido a complexidade inerente a este demandar mais tempo de estudo do código legado. Além disso, estão sendo desenvolvidas novas regras para a previsão, portanto a implementação da previsão foi deixada fora desse trabalho.

5 CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho, foi feita uma reengenharia das aplicações Web do Sistema Irriga, sendo desenvolvido um protótipo funcional com maior segurança, acessibilidade e padronização. No decorrer do desenvolvimento desse trabalho, foi possível verificar na prática a importância da utilização de padrões de projeto de fácil compreensão, e que permitam a redução da base de código.

O *software* desenvolvido nesse trabalho servirá como base para o desenvolvimento de uma completa nova versão do Sistema Irriga, fazendo uso de ferramentas adequadas para o processo (sistemas de controle de versionamento, rastreamento de *bugs* e requisições de mudanças, *wiki* contendo documentação para usuários internos e desenvolvedores), que foram utilizadas no desenvolvimento deste trabalho. O conteúdo do *wiki* que está relacionado ao trabalho está incluído nos anexos.

Como proposta para futuros trabalhos, sugere-se:

- Tornar o cálculo de irrigação e de previsão programáveis, em vez de codificado diretamente na aplicação;
- Adição de *widgets* à biblioteca *Formsmart*, e disponibilizar ao público uma versão utilizando alguma licença livre;
- Melhorar o mecanismo de reconhecimento de linguagem do gerador de código, criando uma gramática apropriada; e
- Efetuar testes de usabilidade do sistema de navegação entre os recursos.

REFERÊNCIAS

CARLESSO, R.; PETRY, M. T.; TROIS, C. Rede de Estações Meteorológicas Automáticas para Prover a Necessidade de Irrigação das Culturas. **Modernización de riegos y uso de tecnologías de la informació**, La Paz, Bolivia, 2007.

ELLISLAB. **CodeIgniter User Guide**. Disponível em: `http://www.codeigniter.com/user_guide`, Acesso em: 30/10/2008.

FIELDING, R.; GETTYS, J.; MOGUL, J.; FRYSTYK, H.; MASINTER, L.; LEACH, P.; BERNERS-LEE, T. **Hypertext Transfer Protocol – HTTP/1.1**. Updated by RFC 2817, RFC 2616 (Draft Standard).

FOUNDATION, F. S. **GNU gettext utilities**. Disponível em: `http://www.gnu.org/software/gettext/manual/gettext.html`, Acesso em: 27/10/2008.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Padrões de projeto**: soluções reutilizáveis de software orientado a objetos. Porto Alegre: [s.n.], 2000.

JACOBS, I.; W3C. **URIs, Addressability, and the use of HTTP GET and POST**. Disponível em: http://www.w3.org/2001/tag/doc/whenToUseGet.html , Acesso em: 02/11/2008.

KORPELA, J. **JavaScript and HTML**: possibilities and caveats. Disponível em: `http://www.cs.tut.fi/~jkorpela/forms/javascript.html`, Acesso em: 26/10/2008.

MICROSOFT CORPORATION. **innerHTML property**. Disponível em: http://msdn.microsoft.com/en-us/library/ms533897(VS.85).aspx, Acesso em: 04/11/2008.

Mozilla Developer Center. **Ajax documentation at the Mozilla Developer Center**. Disponível em: https://developer.mozilla.org/En/AJAX, Acesso em: 04/11/2008.

MOZILLA FOUNDATION. **About Javascript**. Disponível em: `http://developer.mozilla.org/en/docs/About_JavaScript`, Acesso em: 28/07/2008.

PHP Documentation Group, T. **PHP Manual - PHP Gettext**. Disponível em: `http://br.php.net/manual/en/book.gettext.php`, Acesso em: 27/10/2008.

POEDIT. **Site do projeto Poedit**. Disponível em: `http://www.poedit.net`, Acesso em: 31/10/2008.

RESIG, J.; jQuery Team. **jQuery**: the write less, do more, javascript library. Disponível em: `http://www.jquery.com`, Acesso em: 26/10/2008.

RESIG, J.; jQuery Team. **jQuery Javascript Library API Documentation**. Disponível em: `http://docs.jquery.com`, Acesso em: 04/11/2008.

SISTEMA IRRIGA. **Site do Projeto Sistema Irriga**. Disponível em: `http://irriga.proj.ufsm.br`, Acesso em: 28/07/2008.

WILLISON, S. **Unobtrusive JavaScript with jQuery**. Disponível em: `http://simonwillison.net/static/2008/xtech/`, Acesso em: 20/09/2008.

APÊNDICE A

A seguir, o conteúdo das páginas de documentação retiradas do *wiki*.

Instruções para fazer uma cópia local do repositório do NovoSI

1. **Criando as Pastas Locais e Importando o Conteúdo do Repositório**

   a. Efetue login via *SSH* no **backend** e abra um *shell*
   b. Crie o diretório *public_html*, se ainda não existir, dentro do seu diretório de usuário:

   ```
   mkdir ~/public_html
   ```

   c. Entre no diretório *public_html* (`cd public_html`) e faça um *checkout* com o comando:

   ```
   svn checkout http://backend/svn/trunk ALGUM_DIRETORIO
   ```

   digitando a senha de acesso ao *Subversion*, se necessário. Este comando criará o diretório ALGUM_DIRETORIO com uma cópia local (`working copy`) do código do repositório.

   a. No arquivo *application/config/config.php*, configure a variável *`$config['base_url']`* para `/~SEU_USUARIO/ALGUM_DIRETORIO`.
   b. Tente acessar no navegador o endereço: `http://backend/~SEU_USUARIO/ALGUM_DIRETORIO`. Se estiver tudo certo, carregará uma página reclamando erro de conexão com o banco e você pode prosseguir para o próximo passo.

2. **Configurando o banco**

   a. Acesse o banco do backend através de um cliente MySQL (ou no *shell*, com o comando *`mysql -uroot -p`*).
   b. Crie um novo banco:

   ```
   CREATE DATABASE nome_do_database;
   ```

   c. Configure o sistema para usar o banco recém criado no arquivo `application/config/database.php`:

   ```
   $db['default']['database'] = "nome_do_database";
   ```

   d. Agora, pode-se usar o `Makefile` existente para criar e inicializar as tabelas no banco. Altere a variável MYSQL no arquivo `application/scripts/Makefile` de acordo com sua configuração de acesso ao banco e rode:

   ```
   cd application/scripts/ && make dbinit
   ```

Codificação de caracteres

A fim de manter o nosso código-fonte portável, compatível, e a sanidade mental de todo mundo, estamos usando Unicode (UTF-8) como codificação de caracteres no banco de dados, e o código-fonte está sendo mantido TODO EM ASCII.

Por favor, não quebrem esse padrão, nenhuma string e nenhum comentário usando acentos, cedilha ou outro caractere que não-ASCII. Manter tudo ASCII facilita bastante tanto para a administração do site, quanto para a portabilidade do código, já que nós usamos sistemas operacionais diferentes.

Indentação

O recuo está sendo feito com 4 espaços. Menos que isso já começa a ficar ilegível, mais que isso fica difícil de botar bastante coisa na tela.

Usando o CodeIgniter

O CodeIgniter é um framework PHP minimalista focado em segurança e desempenho, que permite bastante flexibilidade no desenvolvimento. A versão que estamos usando atualmente é a ~~1.6.1, 1.6.2~~ 1.6.3, cuja documentação está disponível em http://backend/~eljunior/cidocs.

Há uma versão traduzida do manual de uma versão anterior (1.5.3) *online* em http://www.codeigniter.com.br/manual/, mas está desatualizada e incompatível em alguns trechos com a versão atual. Portanto, cuidado com o que lerem por lá. Recomenda-se a leitura da original em inglês acompanhada de café com bolinhos.

Dicas curtas

Para um relatório detalhado sobre a execução e o desempenho, use no controlador:

```
<?php
$this->output->enable_profiler(TRUE);
?>
```

Modelo de Controller de acesso restrito

Havendo um servico com o nome nomeservico do tipo tiposervico este deve estar disponível na url /tiposervico/nomeservico. Para isso, dentro do diretório application/controllers /tiposervico deve estar um arquivo nomeservico.php um conteúdo semelhante a isto:

```
<?php
class NomeServico extends Controller {
    function NomeServico()
    {
        parent::Controller(); // chama o construtor da classe Controller
        // configura a sessao (idioma, e autenticacao)
        require_permission();
        // carrega os helpers, plugins, modelos, libraries, etc.
        $this->load->helper('file');
        // descomentar para informacoes de depuracao e benchmarking
        // $this->output->enable_profiler(TRUE);
    }
    function index()
    {
        // pagina principal mostrando um Hello from Servico
        $this->template->init_adm();
        $this->template->add_block("Hello from NomeServico");
        $this->template->render();
    }
    // deixar publicos somente os metodos que geram paginas...
    function otra_pagina()
    {
        // ATENCAO: NAO gerar saida direto pra o browser daqui (echo/prin
        // exceto para testes eventuais. as saidas devem ser feitas TODAS
        // senão avacalha com o profiler. em vez de echo, pode-se usar
        // $this->template->add_block("bota aqui tua mensage");
        $this->template->init_adm();
        $this->template->add_view('tipo_servico/servico/otra_pagina',
                                  array('hello' => 'Fala, bundachuja!'));
        $this->template->render();
    }
    // metodos que nao serao paginas devem ser prefixados com underscore
    function _form_config()
    {
         return array(
                    'campo1' => array('rules' => 'required',
                                      'label' => _('Field 1')),
                    'campo2' => array('rules' => 'required',
                                      'label' => _('Field 2')),);
    }
}
```

Os métodos otra_pagina e _form_config estão só para demonstração, só o que é requerido é

o index, e o construtor chamando *parent::Controller();* e *require_permissions($this);*

GERANDO UM NOVO CADASTRO

1. Introdução

O padrão que está sendo utilizado no desenvolvimento deste sistema é conhecido como padrão *MVC* (*Model-View-Controller*) que tem por objetivo básico separar a lógica do sistema da sua apresentação.

Neste modelo, um **componente de visualização** (***view***) renderiza o conteúdo de uma parte particular do modelo e encaminha para o conrolador as ações do usuário. Ele também acessa os dados do modelo via controlador e define como esses dados devem ser apresentados.

O **componente de controle** (***controller***) define o comportamento da aplicação. É ele que interpreta as ações do usuário (como o clique em botões ou seleção de menus) e as mapeia para chamadas do modelo. Com base na ação do usuário e no resultado do processamento do modelo, o controlador seleciona uma visualização a ser exibida como parte da resposta à solicitação do usuário. Há um controlador para cada conjunto de funcionalidades relacionadas.

O **componente de modelo** (***model***) se preocupa apenas com o armazenamento, manipulação e geração dos dados. É um encapsulamento de dados e de comportamento independente da apresentação.

Em resumo:

A ***view*** não se preocupa em como ou onde a informação foi obtida, apenas exibe a informação.

A ***model*** modela os dados e é responsável por tudo que a aplicação vai fazer.

E, finalmente, o ***controller*** determina o fluxo da apresentação, servindo como uma camada intermediária entre a *view* e o *model*.

---

2. Cadastrando um Novo Item no Sistema

Há dois tipos de cadastro a ser realizado no sistema, sendo um deles bem simples, que pode ser gerado automaticamente, e um segundo tipo que envolve mais especificações e deve ser gerado manualmente, seguindo o padrão adotado. As duas formas de se gerar cadastros serão descritas a seguir:

2.1 Geração de Cadastros Simples (automático)

COMPLETAR

2.2 Geração de Cadastros Complexos (manual)

Cadastros deste tipo deverão ser criados seguindo o passo a passo abaixo:

1. No diretório controllers, na subpasta do tipo de serviço ao qual o cadastro se refere, criar um arquivo com os métodos de controle da aplicação. Esse arquivo segue um padrão bem

simples já mostado em http://backend/trac/wiki/ControllerRestritoModelo.

2. No diretório models, criar o arquivo que contém os métodos de acesso ao banco de dados seguindo o padrão abaixo:

COMPLETAR

3. E, finalmente, no diretório views, na subpasta do tipo de serviço ao qual o cadastro se refere, criar os arquivos que contêm os comandos html de apresentação dos resultados enviados pelo controlador. Cada página de visualização terá uma "view" própria. Abaixo é mostrado um exemplo de view comumente utilizada:

COMPLETAR

Geração de cadastro usando gerador de código

Passo 1

A primeira coisa a ser feita é a descrição da tabela no arquivo *scripts/create_tables.sql.in*. Segue um exemplo:

```
TABLE:cultura
cultura VARCHAR(15) NOT NULL
fonte_kc VARCHAR(10) NOT NULL DEFAULT 'tabela'
metodo_acum VARCHAR(10) NOT NULL DEFAULT 'gda'
tbase DECIMAL(4,2) NOT NULL DEFAULT '10.0'
adiamento TINYINT(1) DEFAULT 1
```

Depois disso, deve-se fazer a sincronização desse arquivo com o banco. Tem duas maneiras:

Maneira Destrutiva

A maneira **DESTRUTIVA** (e mais simples) é rodar simplesmente:

```
$ make dbinit     # maneira destrutiva!
```

dentro do diretório *scripts*. Isso vai executar um *DROP* do banco atual, e recriá-lo com os dados nos arquivos *scripts/create-tables.sql* (auto-gerado) e *scripts/init.sql*, então nem sempre é uma boa opção.

Maneira Não-Destrutiva

A maneira não-destrutiva é: dentro do diretório *scripts* rode:

```
  $ make create_tables.sql   # maneira nao destrutiva
```

Esse comando irá atualizar o arquivo *scripts/create_tables.sql* para conter o SQL para criação da tabela descrita. Localize nesse arquivo o código SQL gerado para a tabela recém configurada, e execute-o num prompt MySQL. E tá feita a sincronização! Barbada, uh?

Passo 2

Adicione uma nova entrada no *array* **$config** no arquivo *scripts/translate-scaffold.php*, que contém a configuração da geração de código PHP para os *controllers*, *models* e *views*.

Exemplo:

```php
<?
// code
$config = array(
    'cultura' => array('servico_tipo' => 'admin', 'servico' => 'crops'),
    // more configs
);
// code
?>
```

Nota: Existem uma série configurações que podem ser usadas nos passos 1 e 2, que precisam ser documentadas!

Passo 3

Agora está na hora de rodar o comando que irá efetivamente gerar os arquivos com código PHP:

```
$ make scaffold
```

Serão exibidas no terminal uma série de mensagens sobre o sucesso da operação, e quais arquivos foram escritos (ou não :).

Se o serviço ainda não estiver cadastrado no banco, cadastre-o usando a interface do sistema ou diretamente no banco na tabela **servico**, e ajuste as permissões para o seu usuário ao serviço.

Pronto, agora é só testar e fazer as modificações que forem necessárias.

Dicas de Configurações

Para alguns tipos de cadastros, queremos que determinados campos sejam preenchidos de outra forma. No caso de uma aba de uma subárea, por exemplo, queremos que o id da subárea seja obtido da sessão.

O método atual para fazer isso, é adicionar uma configuração no arquivo de entrada do gerador (correntemente, *create_tables.sql.in*) junto à definição do link da tabela, assim:

```
TABLE:pivo_sub_ajuste:[{"table":"pivo_sub","getfrom":"session"}]
...
```

O par de valores JSON **"getfrom":"session"** fará com que o código PHP gerado gerado pelo controlador exija um id *pivo_sub_id* na sessão, que seja um **id** na tabela *pivo_sub*, e usará esse **id** para filtrar/inserir os dados da tabela *pivo_sub_ajuste*.

> Agora eu tava pensando, acho que seria melhor definir esse tipo de configuração no arquivo de geração de scaffold (*translate-scaffold.php*), e não no arquivo de entrada pra o gerador. (eljunior)