Documentos

ISSN 1517 - 5111
ISSN online 2176-5081
Junho, 2009

260

Framework Orientado a Objetos Aplicado ao Desenvolvimento do IrrigaCerrado

Embrapa

ISSN 1517-5111
ISSN online 2176-5081
Junho, 2009

Empresa Brasileira de Pesquisa Agropecuária
Embrapa Cerrados
Ministério da Agricultura, Pecuária e Abastecimento

Documentos 260

Framework Orientado a Objetos Aplicado ao Desenvolvimento do IrrigaCerrado

Danilo Santos Teodoro
Ozanival Dario Dantas
Luís Gustavo Barioni
Euzebio Medrado da Silva
Daniela Henrique de Oliveira Duarte

Embrapa Cerrados
Planaltina, DF
2009

Exemplares desta publicação podem ser adquiridos na:

Embrapa Cerrados
BR 020, Km 18, Rod. Brasília/Fortaleza
Caixa Postal 08223
CEP 73310-970 Planaltina, DF
Fone: (61) 3388-9898
Fax: (61) 3388-9879
http://www.cpac.embrapa.br
sac@cpac.embrapa.br

Supervisão editorial: *Jussara Flores de Oliveira Arbués*
Equipe de revisão: *Francisca Elijani do Nascimento*
Jussara Flores de Oliveira Arbués
Assistente de revisão: *Elizelva de Carvalho Menezes*
Normalização bibliográfica: *Shirley da Luz Soares Araújo*
Editoração eletrônica: *Leila Sandra Gomes Alencar*
Capa: *Leila Sandra Gomes Alencar*
Impressão e acabamento: *Divino Batista de Sousa*
Alexandre Moreira Veloso

1ª edição
1ª impressão (2009): tiragem 100 exemplares
Edição online (2009)

Dados Internacionais de Catalogação na Publicação (CIP)
Embrapa Cerrados

F813 *Framework* orientado a objetos aplicado ao desenvolvimento do IrrigaCerrado/ Teodoro, Danilo Santos... [et al.]. – Planaltina, DF : Embrapa Cerrados, 2009.
33 p.— (Documentos / Embrapa Cerrados, ISSN 1517-5111, ISSN online 2176-5081 ; 260).

1. Software – irrigação. 2. Cerrado. II. Santos, Teodoro Danilo. III. Série.

005.3 - CDD 21

Autores

Danilo Santos Teodoro
Arquiteto de Sistemas da Eficiência TI
danilo@eficienciati.com.br

Ozanival Dario Dantas
Cientista da Computação, M.Sc.
Analista da Embrapa Cerrados
dario@cpac.embrapa.br

Luís Gustavo Barioni
Engenheiro Agrônomo, Ph.D.
Pesquisador da Embrapa Cerrados
barioni@cpac.embrapa.br

Euzebio Medrado da Silva
Engenheiro Agrônomo, Ph.D.
Pesquisador da Embrapa Cerrados
euzebio.medrado@gmail.com

Daniela Henrique de Oliveira Duarte
Tecnóloga em Processamento de Dados
Analista da Embrapa Cerrados
daniela@cpac.embrapa.br

Agradecimentos

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq - Brasil pelo apoio financeiro ao projeto Economia Espacial no Uso e Conflito da Água (Processo 555952/2006-1) do Programa de Revitalização da Bacia Hidrográfica do Rio São Francisco.

Apresentação

A área de tecnologia da informação vem procurando novas técnicas para padronizar e facilitar o desenvolvimento de sistemas. Nesse sentido, foi criado o conceito de *frameworks*, que é a criação de uma estrutura de classes relacionadas, que constitui uma implementação apenas iniciada para atender um conjunto de aplicações de um determinado domínio. A partir dessa estrutura inicial, podem-se incluir novas classes herdadas das já existentes, com objetivo de atender às necessidades específicas do domínio do negócio, nesse caso, os sistemas de irrigação.

No desenvolvimento do software IrrigaCerrados, foi utilizada uma *framework* que possibilitasse a generalidade, a flexibilidade e a extensibilidade das classes. Com isso, ganhou-se: rapidez no desenvolvimento; uma estrutura flexível para atender os diversos tipos de sistemas de irrigação; e a possibilidade de expansão para comportar novos sistemas de irrigação.

José Robson Bezerra Sereno
Chefe-Geral da Embrapa Cerrados

Sumário

Framework Orientado a Objetos Aplicado ao Desenvolvimento do IrrigaCerrado[1]

Danilo Santos Teodoro
Ozanival Dario Dantas
Luís Gustavo Barioni
Euzébio Medrado da Silva
Daniela Henrique de Oliveira Duarte

Introdução

O IrrigaCerrado é um software, em desenvolvimento, que nasceu de um projeto de pesquisa cujo objetivo é avaliar e dimensionar, de forma otimizada, o uso da água no manejo da irrigação. Para tanto, deve realizar as seguintes tarefas: avaliação do sistema de irrigação (seja ele, pivô convencional, pivô lepa, gotejamento, aspersão convencional, aspersão sob copa, etc.); avaliação do solo (utilizando tensiometria); e manejo da irrigação (respondendo às perguntas: "quando irrigar?" e "quanto irrigar?").

De acordo com as tarefas descritas, o software deveria possuir uma estrutura composta por cadastros (da propriedade, de equipamentos de irrigação, de coletas e resultados de avaliações, tanto da irrigação quanto do solo, de culturas, entre outros) e por funções para realizar cálculos (área relativa individual, área total por grupo, lâmina medida, área adequadamente irrigada, etc.). Adicionalmente, todas as informações geradas no sistema deveriam ser armazenas em um banco de dados.

[1] Software desenvolvido na Embrapa Cerrados, para uso na irrigação, que busca responder as perguntas: "quando irrigar" e "quanto irrigar".

Ao analisar a estrutura necessária para o software, foi percebida a ocorrência de entidades semelhantes (entre cadastros, avaliações, conexões entre os módulos, funções, etc.), o que permitiu a definição de padrões. A definição de padrões é importante para permitir a reutilização dos módulos, facilitar o entendimento do projeto, melhorar sua manutenibilidade e extensibilidade e, finalmente, fundamentar uma base sólida para o desenvolvimento do software (LOBO, 2009).

Dessa forma, a possibilidade de se padronizar grande parte do projeto IrrigaCerrado permitiu a criação de uma *framework*[2], a fim de se estabelecer uma base para a construção do software e de outros aplicativos com arquitetura física do tipo "cliente x servidor". Com o objetivo de realizar uma separação entre a apresentação do sistema e a lógica de negócio e controle, a *framework* foi divida em três camadas: modelo (contém os dados relativos ao domínio), visão (tela de apresentação e interação com o usuário) e controlador (define a forma que a interface reage à entrada do usuário e fluxo do sistema), também conhecido como arquitetura MVC (modelo-visão-controlador) (GAMMA, E. et al., 1995).

Arquitetura

Para facilitar a interoperabilidade de componentes e a separação dos módulos ("dividir para conquistar"), foi adotado o modelo em camadas (KRUCHTEN, 2003). Baseado nessa sentença, existe um princípio que diz:

1. Não conecte ou acople objetos que não são Interface do usuário (IU) diretamente a objetos de IU. Por exemplo, não faça um objeto de software Venda (um objeto do "domínio" não-IU) ter referência a um objeto janela. Por quê? Porque as janelas estão relacionadas a uma aplicação em particular, enquanto (idealmente) os objetos não-janela podem ser reusados em novas aplicações ou ligados a uma nova interface.

[2] Conjunto de classes que cooperam entre si e compõem um projeto reutilizável para uma categoria específica de software (GAMMA, et al., 1995).

2. Não coloque a lógica da aplicação (como um cálculo de imposto) em métodos de objetos de IU. Objetos de IU devem apenas iniciar elementos IU, receber eventos de IU (como um clique de mouse em um botão) e delegar solicitações de lógica da aplicação a objetos não-IU (tais como, objetos de domínio) (LARMAN, 2002).

Modelo em Camadas

O modelo adotado está dividido em três camadas: Modelo, Visão e Controlador. A grande vantagem desse tipo de modelo é a possibilidade de realizar diferentes implementações de componentes para cada uma das camadas, tornando-as permutáveis. Com isso, consegue-se, ao mesmo tempo, uma interface padrão para basear o aplicativo e flexibilidade para fornecer uma implementação totalmente diferenciada, que ainda assim se "encaixará" com o sistema (LOBO, 2009). Daqui para frente, essa arquitetura será chamada apenas de MVC, de acordo com a Fig. 1.

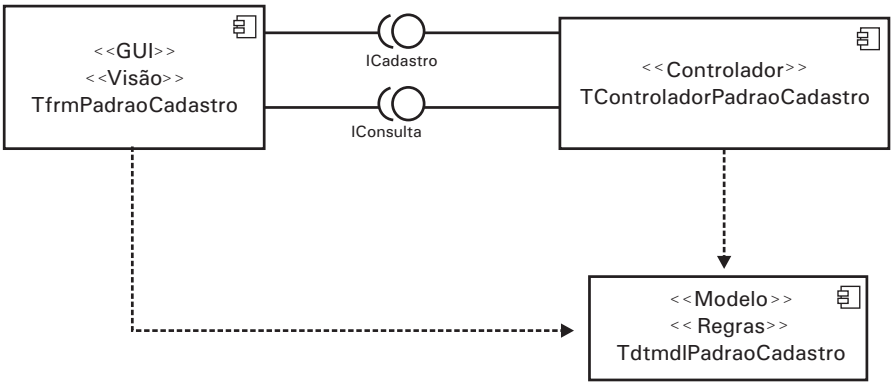

Fig. 1. Componentes do MVC.

Camada de visão

Camada responsável pela interface gráfica, que dará acesso às funcionalidades do sistema. Representada por um descendente de TfrmPadraoCadastro ou até mesmo de TForm. Por padrão, permite consultar e cadastrar informações, baseado no controlador associado.

Além das ações padrão de consulta e cadastro, pode acessar diretamente o controlador para ter acesso a serviços diferenciados e específicos da tela atual. Devido à infraestrutura Delphi, a Visão deve enxergar componentes do Modelo para ligá-los aos seus componentes visuais, mas ela nunca deve chamar os serviços do modelo diretamente, devendo para isso passar pelo Controlador, como em:

- Visão (Form): ContratoCadastro.GravarAlteracao.
- Controlador (ICadastro): ModeloCadastro.GravarAlteracao.
- Modelo (DataModule): cIntdtstCadastro.ApplyUpdates(0).

Na Fig. 2, mostram-se as classes que compõem a camada de visão do framework.

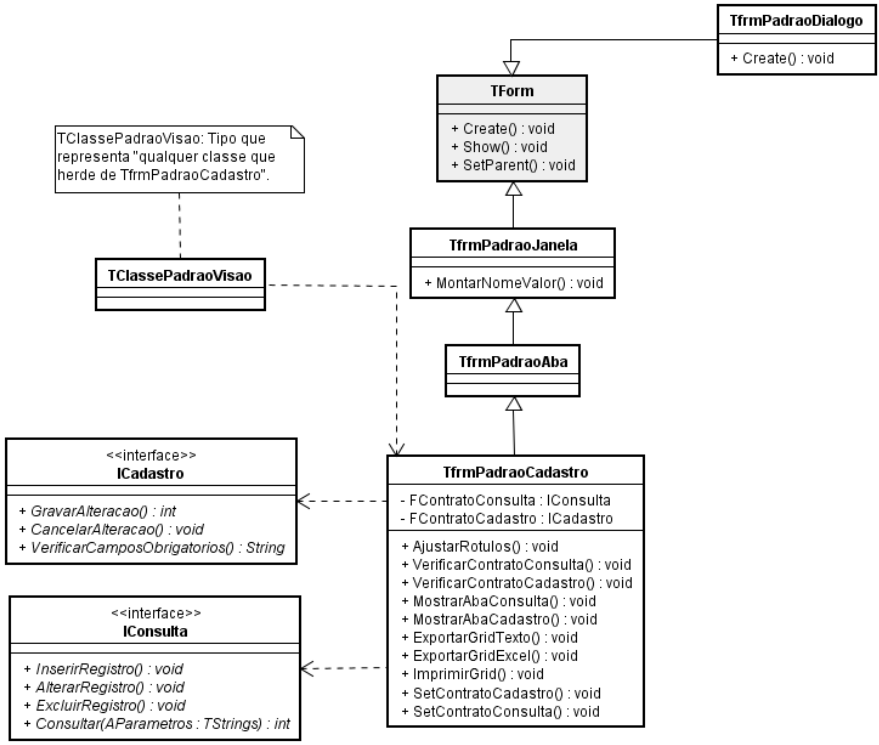

Fig. 2. Classes que compõem a camada "Visão".

Essa hierarquia descreve as principais classes que compõem a camada "Visão". A classe "TForm" é a única que não foi personalizada.

- TForm: classe padrão de janelas no Delphi, de onde se deve herdar somente em último caso, em que não há nenhuma janela compatível para herança.

- TfrmPadraoDialogo: janela independente, geralmente aparece de forma "modal" (exclusiva). Não possui vínculo com controladores, mas pode possuir. Não é contida dentro de um PageControl. Utilizada quando é necessário "perguntar" algo ao usuário, pedir feedback ou qualquer operação exclusiva.

- TfrmPadraoJanela: tela padrão para todas as janelas no sistema. Basicamente define a aparência das janelas, por isso não possui barra de título, já que deverá estar contida dentro da aba de um PageControl.

- TfrmPadraoAba: janela que possui um PageControl, devendo ser a tela padrão (direta ou indiretamente) de toda visão que precisar de abas (como consulta e cadastro, por exemplo).

- TfrmPadraoCadastro: classe mais importante da camada de visão, define o comportamento e aparência da maioria das janelas. Permite consulta e cadastro graças a sua associação com as interfaces IConsulta e ICadastro. Por padrão, quem implementa essas interfaces é o controlador. Implementa vários métodos utilizados para cadastro e consulta, além de rotinas de exportação e impressão. Essa classe deve ser utilizada sempre que uma visão que permita consulta e cadastro se fizer necessária. Na Fig. 3, mostra-se a aparência da tela de consulta e, na Fig. 4, exibe-se a tela de cadastro dessa classe.

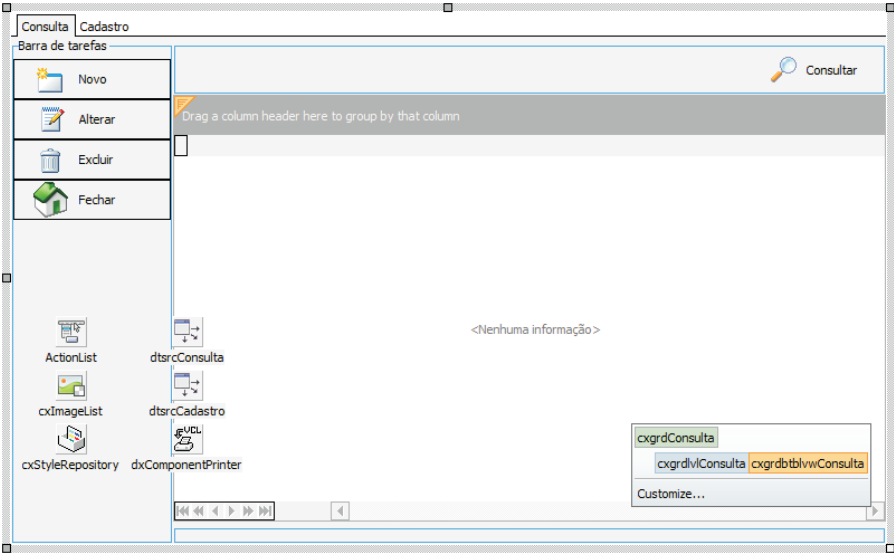

Fig. 3. TfrmPadraoCadastro (aba consulta).

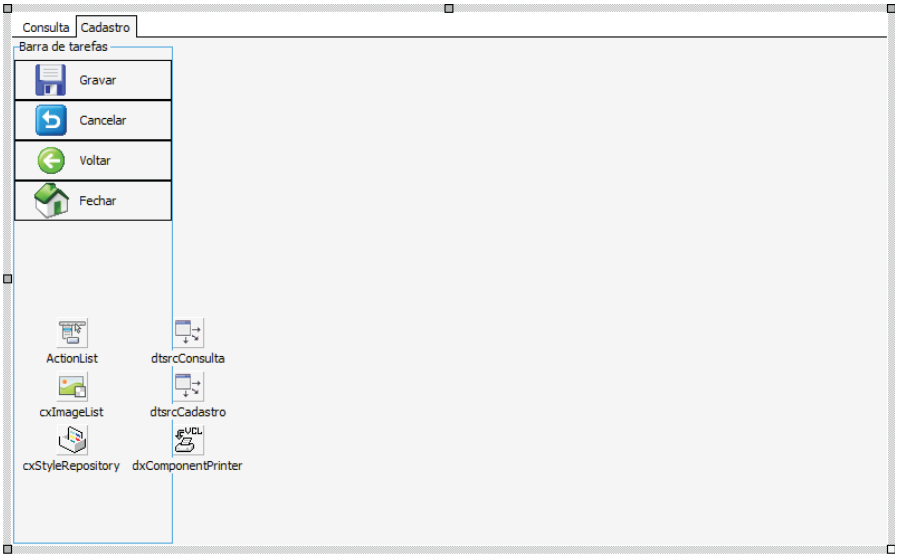

Fig. 4. TfrmPadraoCadastro (aba cadastro).

Camada de controle

Essa é a camada que deve conhecer as outras, servindo como intermediador entre visão e modelo. Representada por um descendente de TControladorPadraoCadastro ou, até mesmo, TObject em casos em que a implementação é muito diferente do restante do projeto. Por padrão, realiza as interfaces de Consulta e Cadastro, permitindo:

- Criar um novo registro.
- Alterar um registro existente.
- Excluir um registro.
- Realizar uma consulta.
- Gravar alterações.
- Cancelar alterações.

Um controlador tem como função:

- Controlar o fluxo do sistema.
- Guardar referência e instanciar a visão e o modelo.
- Encaminhar chamadas da visão ao modelo, quando necessário.
- Realizar algum código que não seja relacionado nem com a apresentação (janela e controles do usuário), nem com os dados propriamente ditos.
- Preparar o local (recipiente) onde a visão será exibida.

Um controlador deve sempre implementar o design pattern Singleton, para permitir uma única instância do controlador no sistema e facilitar o acesso a ela (sempre deve ser acessado usando <NomeDaClasse>.ObterInstancia.<NomeDoMétodo>, como em:

TCategoria.ObterInstancia.MostrarConsulta;

Na Fig. 5, mostra-se a hierarquia das principais classes da camada de controle (Controlador).

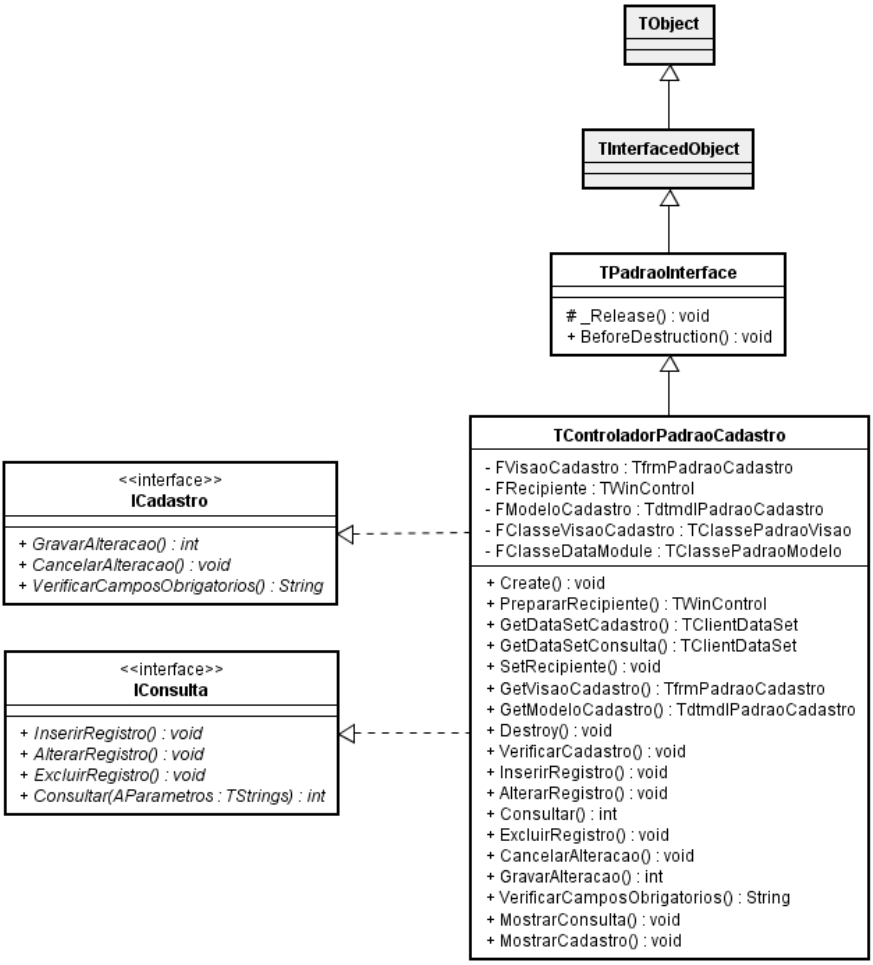

Fig. 5. Classes que compõe a camada "Controlador".

As classes “TObject” e “TInterfacedObject” são nativas do Delphi. A seguir a descrição da hierarquia:

- TObject: essa é a classe ancestral de todas as classes. Deve-se herdar dela somente quando não é necessário implementar uma interface e quando não há outra classe mais completa compatível.

- TInterfacedObject: classe do Delphi para ser herdada quando uma dada classe implementa alguma interfaces.
- TPadraoInterface: altera o comportamento da classe TInterfacedObject para evitar algumas questões com classes singleton e visões que utilizam interfaces. Usada internamente.
- IConsulta: interface representando as operações comuns em uma tela de pesquisa. Qualquer classe que implementar/realizar essa interface pode ser conectada a um descendente de TfrmPadraoCadastro.
- ICadastro: interface contendo as operações comuns em uma tela de edição. Qualquer classe que implementar/realizar ICadastro pode ser conectada a um descendente de TfrmPadraoCadastro.
- TControladorPadraoCadastro: classe mais importante da hierarquia de controladores. Realiza as interfaces ICadastro e IConsulta, gerencia (referencia, instancia, controla e acessa) sua própria visão e modelo, controla o fluxo da aplicação, etc. Deve ser utilizada sempre que o módulo em questão possuir o seguinte comportamento: visão herda (direta ou indiretamente) de TfrmPadraoCadastro, modelo herda (direta ou indiretamente) de TdtmdlPadraoCadastro, contém consulta e cadastro.

Camada de modelo

Representa os dados e geralmente contém as regras de negócio – que também podem estar na base de dados. Os dados correntes no modelo são visualizados através da visão, às vezes em forma de consulta, às vezes em forma de cadastro, entre outros. É representada por um descendente de TdtmdlPadraoCadastro, ou até mesmo TDataModule, quando os componentes variam muito do padrão "consulta & cadastro". Por padrão, implementa os seguintes serviços:

- Inserir registro.
- Editar registro selecionado.
- Excluir registro selecionado.

- Gravar alteração.
- Cancelar alteração.
- Consultar.

A diferença desses serviços com os serviços do controlador é que, enquanto o modelo realiza sua implementação diretamente nos dados, o controlador apenas encaminha as chamadas aos respectivos serviços do modelo, com algumas poucas alterações na visão.

Na Fig. 6, mostram-se as principais classes que compõem a camada de Modelo.

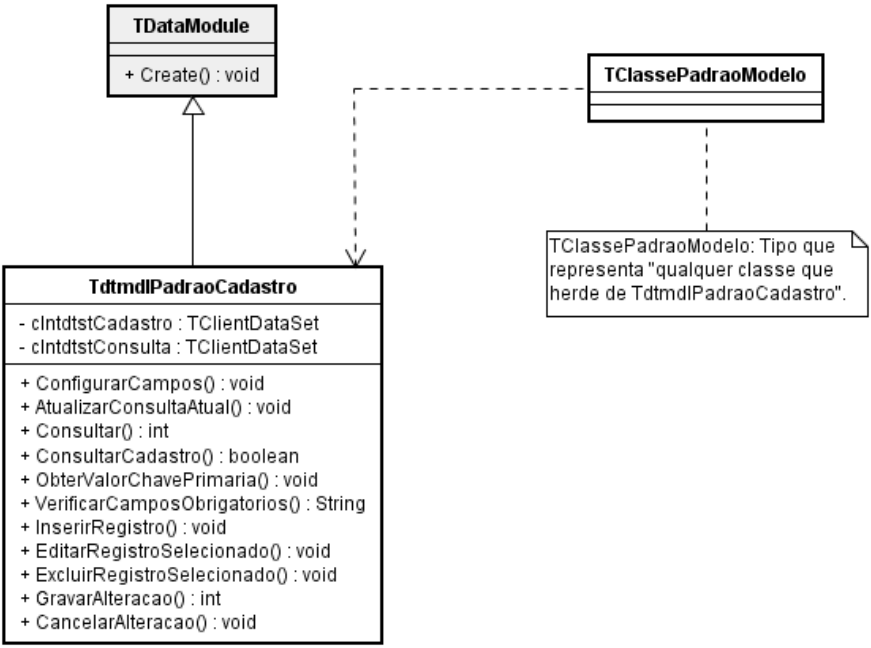

Fig. 6. Classes que compõe a camada "Modelo".

A classe “TDataModule”, nativa do Delhi, é a única que não foi personalizada:

- TDataModule: repositório de dados "default" do Delphi, deve ser utilizado apenas quando não houver uma classe descendente que seja compatível. Deve ser usada quando o módulo a ser desenvolvido não seguir o padrão dos demais, ou seja, se o módulo em questão não possuir uma lista e um cadastro. Essa classe, bem como todas as suas descendentes, é responsável pela comunicação junto ao SGBD e pelas regras de negócio.
- TdtmdlPadraoCadastro: classe mais importante da hierarquia de Modelos. Contém componentes de acesso aos dados para consulta e cadastro; e codificação necessária para a realização de consultas, inserção, alteração, exclusão de registros, gravação, entre outros. Sabe informar se há algum campo obrigatório não preenchido. Pode implementar serviços para cálculos e repositórios de regras de negócio, que serão chamados pelo respectivo controlador.

Framework Aplicado ao IrrigaCerrado

Com base no *framework* descrito anteriormente, o projeto IrrigaCerrado será implementado, utilizando como exemplo o módulo avaliação de pivô convencional.

Camada de visão com o pivô convencional

Na Fig. 7, mostra-se o relacionamento da camada de apresentação da avaliação de Pivô Convencional com o framework. Como existem várias avaliações parecidas, foi definida a interface IAvaliacao, a qual é realizada pela nova classe TfrmPadraoAvaliacaoSistemaIrriga, que serve como "ancestral" para todas as avaliações do sistema. As avaliações possuem categorias – como Pivô Central, Localizada, Aspersão –, por isso foi definida uma classe padrão para cada categoria (como TfrmPadraoAvaliacaoPivoCentral, por exemplo) com o intuito de facilitar as futuras manutenções do software. A classe TfrmAvaliacaoPivoConvencional representa a visão do módulo.

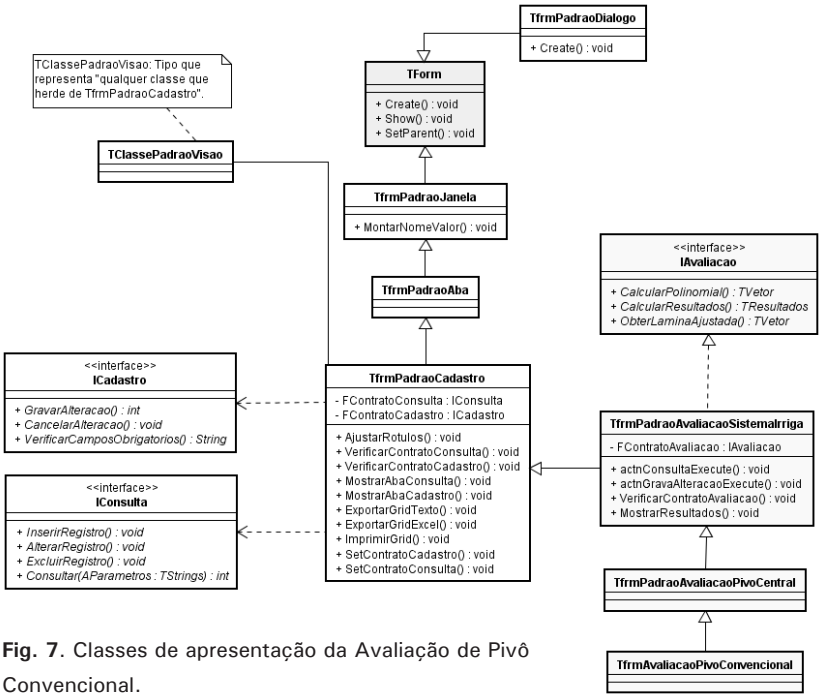

Fig. 7. Classes de apresentação da Avaliação de Pivô Convencional.

A seguir são mostradas as várias abas da tela Avaliação de Pivô Convencional (Fig. 8 a 11).

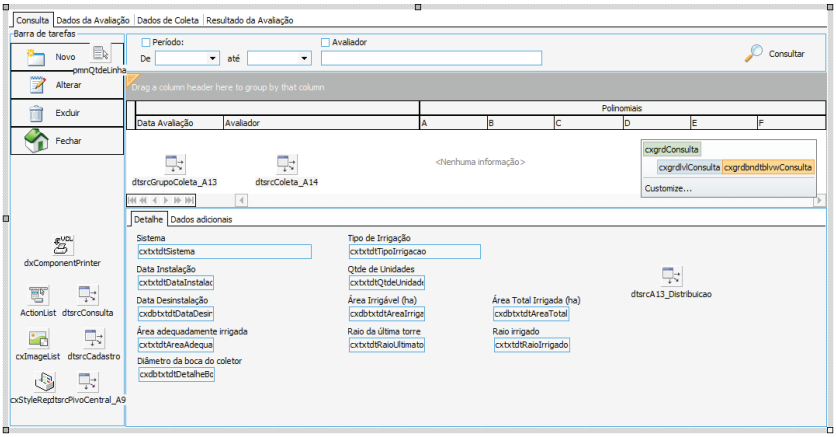

Fig. 8. Aba de consulta do módulo Avaliação de Pivô Convencional.

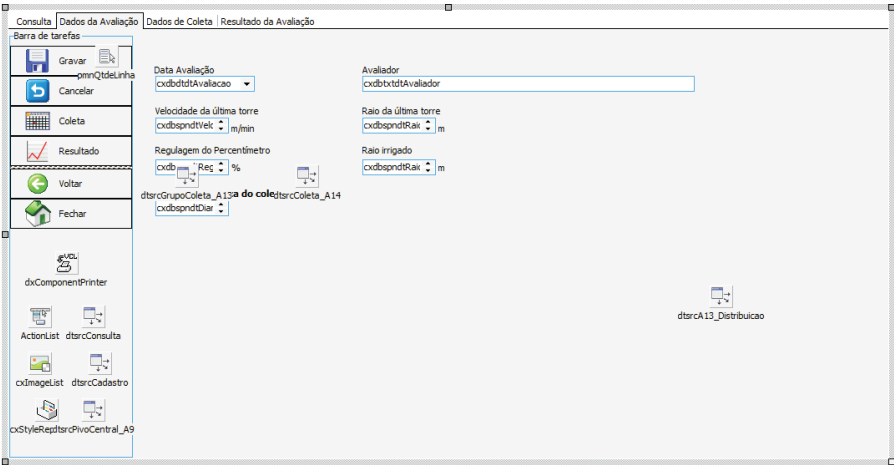

Fig. 9. Aba de cadastro do módulo Avaliação de Pivô Convencional.

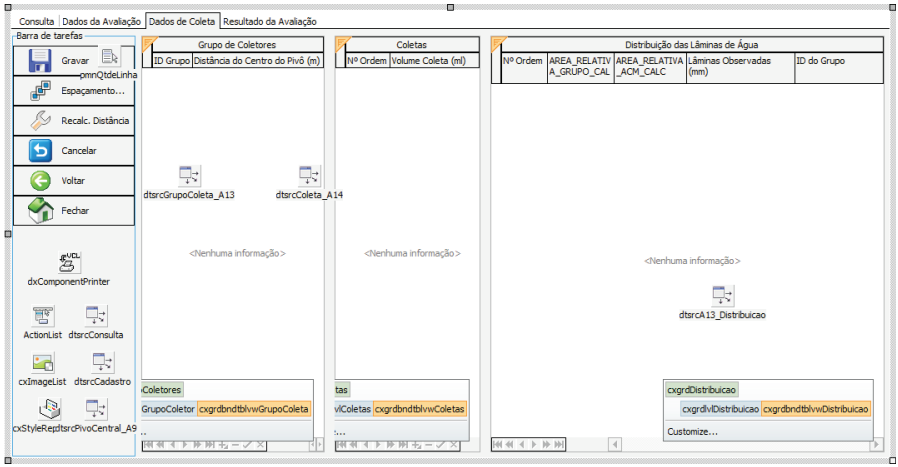

Fig. 10. Aba de coleta do módulo Avaliação de Pivô Convencional.

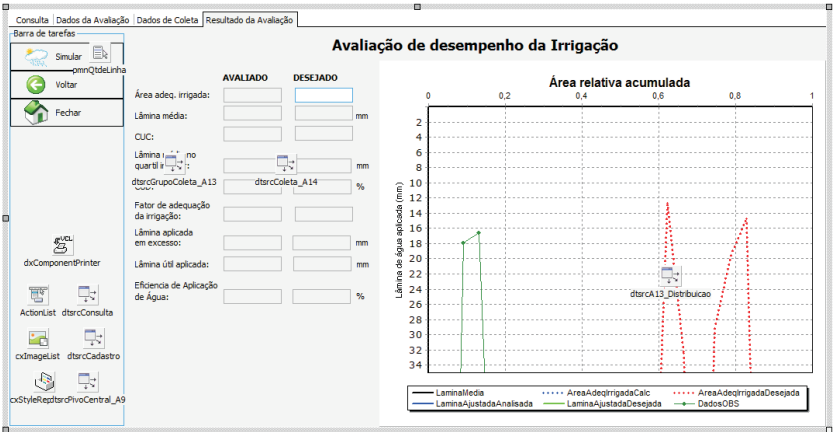

Fig. 11. Aba de resultados do módulo Avaliação de Pivô Convencional.

Camada de controle com o pivô convencional

As classes destacadas em amarelo, na Fig. 12, representam a herança necessária para desenvolver o controlador referente ao módulo Avaliação de Pivô Convencional.

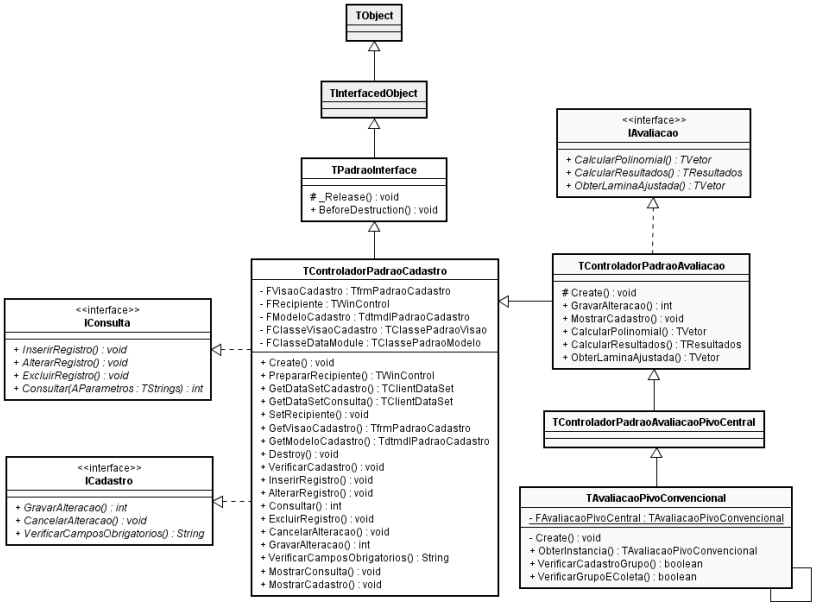

Fig. 12. Classes do controlador da Avaliação de Pivô Convencional.

Camada de modelo com o pivô convencional

As classes destacadas em amarelo, na Fig. 13, representam a herança necessária para desenvolver o modelo referente ao módulo Avaliação de Pivô Convencional:

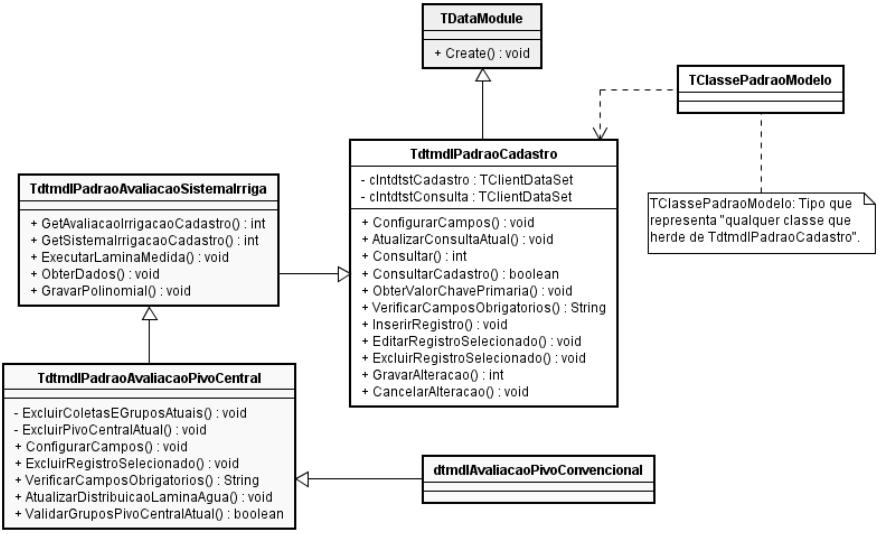

Fig. 13. Classes do modelo da Avaliação de Pivô Convencional.

Entendendo o Funcionamento das Camadas no IrrigaCerrado

A comunicação entre as camadas é realizada pelo controlador. Quando o usuário executa uma ação em uma tela qualquer – como abertura de um módulo, consulta ou gravação –, caso essa informação não possa ser tratada pela própria visão, é enviada uma mensagem ao respectivo controlador para que ele resolva o problema. O controlador, cujo papel é determinar o fluxo da aplicação, identifica se pode resolver o problema ou passá-lo para outra classe ou camada. Se for algo relacionado aos dados, é enviada a devida mensagem ao modelo associado ao controlador, que, em última instância, executa a ação – como consultar um registro ou persistir uma informação. No exemplo da Fig. 14, é demonstrada uma situação em que o usuário requisita

o módulo "Propriedade", fazendo com que o controlador decida se é necessário instanciar a visão e o modelo, para depois invocar um serviço que realize uma consulta. Devido à forma de conexão (*binding*) do Delphi, cada alteração no modelo é refletida na visão, fazendo com que o usuário veja as informações, que são apresentadas em um *grid*.

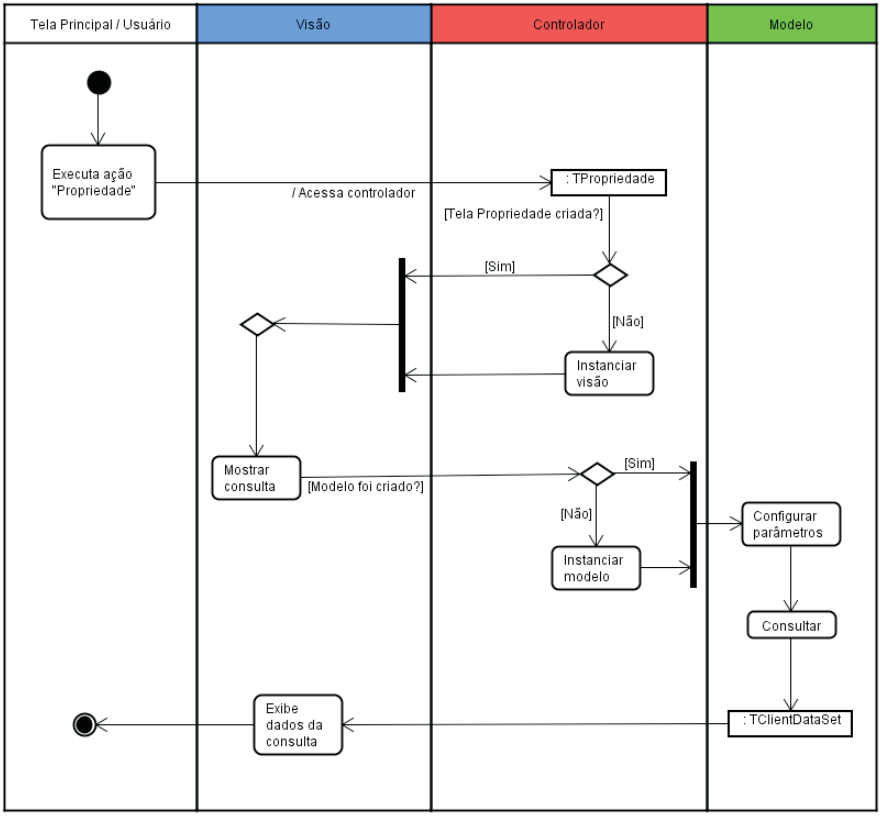

Fig. 14. Diagrama de atividades mostrando o relacionamento entre camadas.

Exibindo um Módulo

No diagrama da Fig. 15, são mostrados o relacionamento e a troca de informações entre objetos com o objetivo de consultar as propriedades cadastradas.

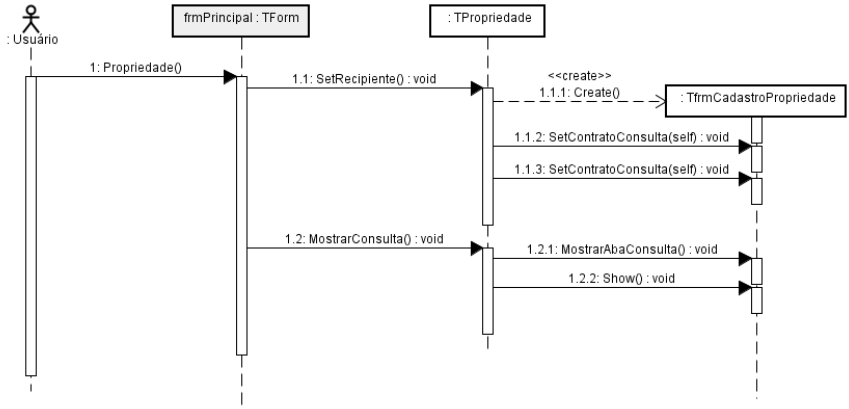

Fig. 15. Mostrar Consulta Propriedade.

Essa é a sequência necessária para exibir a consulta do módulo "Propriedade". Inicialmente, o pesquisador, pela tela principal (frmPrincipal), clica na opção "Propriedade". Com isso, o frmPrincipal invoca o método "SetRecipiente", que trata de criar visão, se necessário, e indica que tanto a interface de Consulta quanto a interface de Cadastro serão implementadas pela instância da classe "TPropriedade" (self). Com isso, se já não existir, é criada uma nova aba na tela principal. O fluxo termina quando o frmPrincipal chama o método "MostrarConsulta" da classe TPropriedade, que manda a visão ativar a aba de consulta e mostrar sua tela (Show).

Realizando uma Consulta

No próximo diagrama (Fig. 16), é demonstrada a interação entre objetos/camadas para a execução de uma consulta.

Após o pesquisador ativar o módulo "Propriedade" e clicar no botão "Consultar" – representado pela ação actnConsultaExecute –, a visão, através da sua superclasse "TfrmPadraoCadastro", verifica se ela pode realizar uma consulta e, em caso positivo, chama o método "Consultar" do objeto responsável pela consulta – o controlador do módulo, TPropriedade – o controlador basicamente encaminha essa chamada de método ao modelo que efetivamente realiza a consulta na base de

dados. Ao fazê-lo, os dados da consulta automaticamente aparecerão na visão (TfrmCadastroPropriedade).

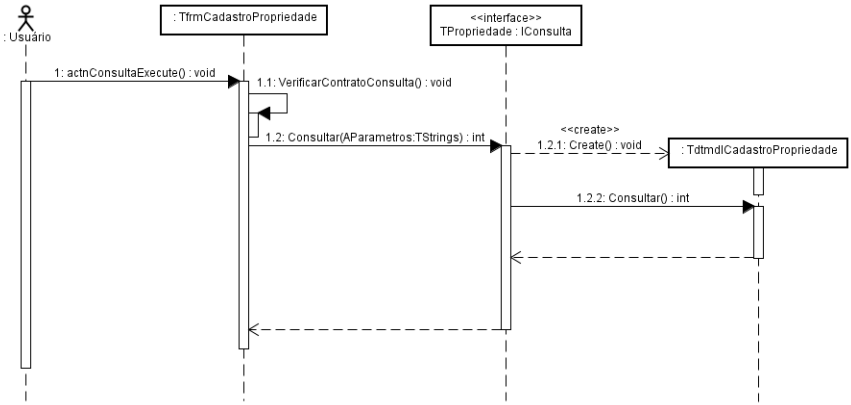

Fig. 16. Consultando uma Propriedade.

Persistindo uma Informação

No diagrama da Fig. 17, mostram-se o relacionamento e a troca de informações entre objetos com o objetivo de gravar as alterações realizadas em uma dada propriedade.

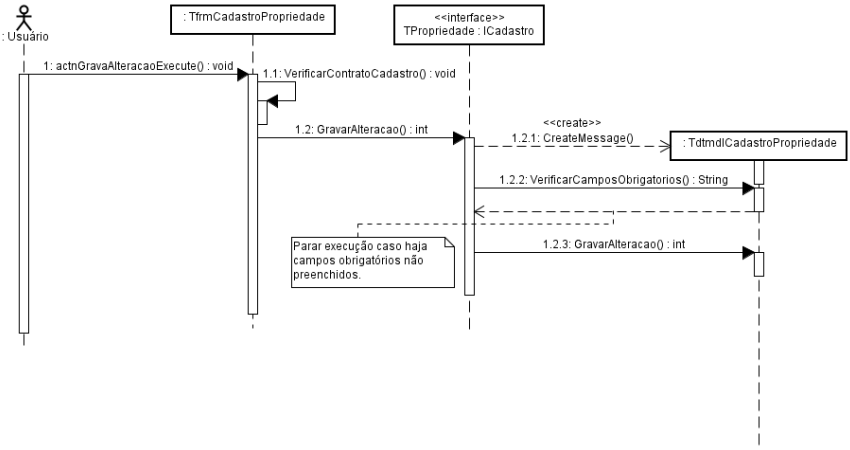

Fig. 17. Gravar Propriedade.

Como na consulta, após o usuário alterar uma informação e clicar no botão "Gravar" (actnGravaAlteracaoExecute), a visão do módulo "Propriedade" verifica se a ela está apta a realizar cadastros (VerificarContratoCadastro). Se estiver, manda o controlador gravar alterações. O controlador cria o modelo, se necessário, e verifica se existe algum campo obrigatório não preenchido. Se houver campos obrigatórios não preenchidos, o controlador exibe a mensagem adequada e pára o fluxo. Se estiver tudo certo, pede para o modelo enviar as alterações ao banco de dados. É possível enxergar a diferença de papéis: o Modelo tem a função de verificar se existem campos obrigatórios não preenchidos, mas quem controla o fluxo – ou seja, se deve parar ou não – é o controlador.

Avaliando um Pivô Convencional

No diagrama da Fig. 18, mostram-se o relacionamento e a troca de informações entre objetos com o objetivo de avaliar um pivô convencional.

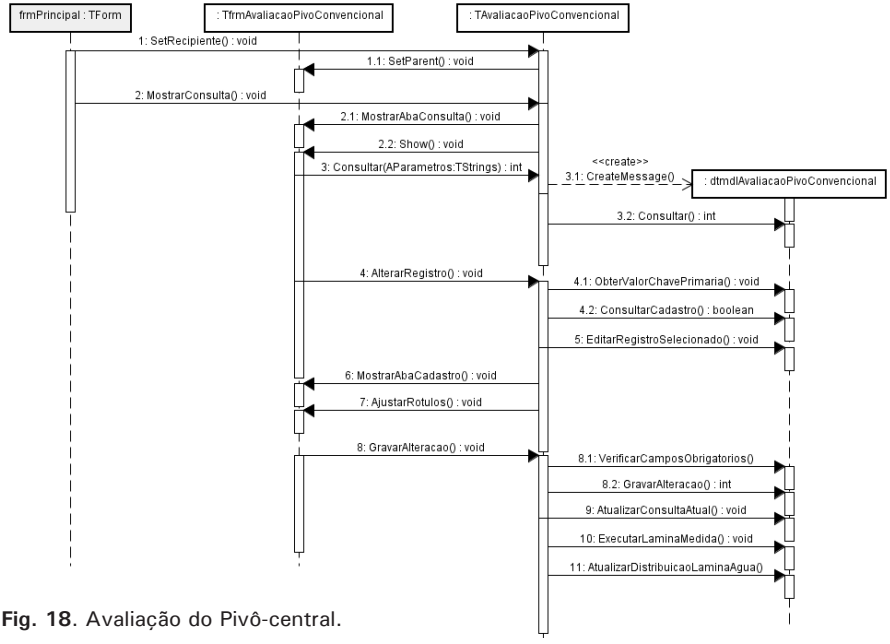

Fig. 18. Avaliação do Pivô-central.

A próxima sequência mostra o ciclo completo de acesso, consulta e alteração de um registro.

1. A partir da tela principal, é pedido ao controlador que encontre um recipiente (parent) para a visão. Note que nem a tela principal sabe quem é a visão, apenas o controlador.

2. Depois de exibir a consulta do módulo "Propriedade", o pesquisador executa a ação "Consultar", que verifica se existe um objeto que implemente a interface IConsulta: se existir, chama seu método "Consultar", passando os parâmetros da consulta (uma lista do tipo: nome_parametro = valor_parametro).

3. Isso faz com que o controlador crie, se necessário, o modelo e chame a função Consultar deste datamodule recém-criado.

4. Com isso, caso a consulta retorne registros, eles serão exibidos na visão (TfrmAvaliacaoPivoCentral).

5. O Pesquisador seleciona um registro e executa a ação "AlterarRegistro" da interface ICadastro – representada pelo controlador TPropriedade.

6. Essa chamada, por padrão, faz com que o controlador recupere os valores que compõe a chave-primária do registro selecionado (ObterValorChavePrimaria).

7. Com esses valores em memória, é chamado o método ConsultarCadastro, que faz uma pesquisa utilizando os valores obtidos acima no componente de cadastro/edição a fim de editar o registro selecionado.

8. Se a consulta retornar um registro, o componente de cadastro entra em modo de edição e o controlador manda uma mensagem para a visão ativar sua aba de cadastro e aparecer na tela.

9. Após alterar alguma informação, o Pesquisador executa a ação "Gravar", que chama o método "GravarAlteracao" do controlador TPropriedade.

10. A instância da classe TPropriedade verifica se existem campos obrigatórios não preenchidos. Se houver algum, o controlador exibe uma mensagem para o Pesquisador e pára o fluxo, do contrário, é chamado o método "GravarAlteracao" do modelo, que persiste as alterações do registro na base de dados.

11. Em seguida, o controlador TPropriedade manda uma mensagem para o modelo atualizar sua consulta.

12. Como houve alterações no pivô-central, é necessário executar alguns procedimentos no SGBD para atualizar alguns valores. Isso é feito quando o controlador manda o modelo executar "LaminaMedida".

13. Para finalizar, a tela de Avaliação de Pivô-central contém um controle que exibe os dados que foram calculados pelo procedimento anterior. Para mostrar os dados atuais, o controlador chama o método AtualizarDistribuicaoLaminaAgua.

Conclusão

A definição de um conjunto de classes padronizadas e documentadas que seguem um modelo consistente como o MVC traz diversos benefícios a um projeto: padronização; facilidade e agilidade na manutenção; melhoria no entendimento do projeto, inclusive para novos membros; o código fica mais legível; reutilização de entidades; habilita o trabalho em equipe, pois os módulos são divididos em, no mínimo, três arquivos.

A partir da presente *framework*, podem ser desenvolvidos diversos tipos de aplicativos que tenham as seguintes características: arquitetura do tipo cliente x servidor; armazenamento e consulta em banco

de dados relacional; desenvolvimento baseado em módulos, como cadastros, consultas, avaliações, etc; seja baseado na linguagem delphi a partir de 2005 (por causa de algumas características da linguagem) desktop.

A ferramenta RAD Studio (Delphi) foi escolhida pela sua produtividade, principalmente em projetos desktop de pequeno e médio porte, cliente/servidor e com acesso a banco de dados. O *framework*, por sua vez, privilegia padronização e rapidez na programação, ou seja, tenta utilizar o melhor da orientação a objetos e a forma rápida de desenvolvimento. A divisão em camadas possibilitou a separação de responsabilidades do código e uma maior facilidade de manutenção.

Referências

GAMMA, E.; JOHNSON, R.; VLISSIDES, J.; HELM, R. **Design Patterns**: elements of reusable object-oriented software. 2. ed. Boston: Addison-Wesley, 1995.

KRUCHTEN, P. **The Rational unified process**: an introduction. Boston: Addison Wesley, 2003.

LARMAN, C. **Applying UML and patterns**. 2. ed. Upper Saddle River: Prentice Hall, 2002.

LOBO, E. J. R. **Guia prático de engenharia de software**. São Paulo: Digerati, 2009.

Object Oriented *Framework* Applied to the Development of IrrigaCerrado Software

Abstract

The IrrigaCerrado software is a product under development to facilitate the irrigation users to decide the moment and the quantity of water to apply based on soil-water measurements. Following the directions given by the irrigation specialist, it was realized that the software would have a structure based on information about properties, irrigation equipments, evaluation of irrigation performance, soil-water measurements and water irrigation planning to guide irrigation users. Furthermore, all generated information should be stored in a database. Based on the analysis of the necessary structure to devise the software, it was realized the occurrence of various similar entities, which could be reorganized by using the technique of designing patterns. This allowed to develop a framework with the objective to establish a base for constructing software with physical architecture of the type client-sever. In this case, the framework was subdivided into three layers: model, which contains the data related to the business domain; vision, graphical interface for users; and controllers, which define the way as the interface interact with the users and how the system must deal with the information given.

Index terms: client-server, three-layer architecture, developing architecture.