Sistema de Lembretes

EP de Programação Orientada a Objetos
Fase 2: Uma interface básica

Ana Paula Oliveira dos Santos
Ariel Martini
David da Silva Pires
Paulo Meirelles

IME-Instituto de Matemática e Estatística
USP-Universidade de São Paulo

Professor: Fábio Kon
Monitor: Hugo Corbucci

28 de maio de 2008

1 Introdução

Esta fase do exercício-programa cuidou do desenvolvimento de uma interface de manipulação do modelo desenvolvido na fase 1.

A implementação desta interface revelou diversas necessidades que não eram satisfeitas pelo modelo anterior, exigindo intervenções em decisões já tomadas. Tais ações completam e iteram um ciclo que é comum na construção de sistemas mais complexos; revelam não apenas um melhor entendimento do domínio por parte dos desenvolvedores como também humildade e maturidade para assumir erros e persistência na resolução dos problemas propostos.

Das opções de funcionalidades que nos foram dadas (vide enunciado, em anexo), fizemos duas:

- Fazer com que a adição e remoção de tarefas, assim como a ordenação, não recarregue a página (AJAX).
- Persistir os dados usando o banco de dados orientado a objetos Magma. Tanto os usuários quanto suas tarefas e garantir segurança (nada de salvar a senha como texto puro).

Este documento está organizado da seguinte forma: na Seção 2 são descritas as novas ferramentas usadas, tanto para programação como para comunicação entre os membros. A Seção 3 trata da modelagem do sistema. Já na Seção 4 explicitamos as classes, seus atributos e métodos. Por fim, na Seção 5 são detalhadas alguns padrões de boas práticas de programação em Smalltalk. Ao final há uma relação das referências bibliográficas usadas no desenvolvimento do sistema e de sua documentação.

O enunciado da fase foi, desta vez enviado como um anexo. Tomamos essa decisão para não dar a impressão da presença de prolixidade no relatório. Salientamos que a presença do enunciado junto ao sistema entregue é importante para futuras referências quanto à especificação do sistema.

O relatório termina com um manifesto referente à dificuldade que os integrantes do grupo encontraram para a execução desta fase.

2 Uso de Novas Ferramentas

Na implementação desta segunda fase, foram usadas algumas novas ferramentas de desenvolvimento:

Squeak Web: Três integrantes do grupo usaram uma versão do *Squeak* voltada para programação com interface web, chamada *Squeak Web*, a mesma apresentada em sala de aula pelo monitor. O outro integrante usou a versão 2.8 do *Seaside* baseada no *Squeak* versão 3.10, divulgada no site do *Seaside* como a versão "*Seaside One-Click Experience*". Apesar dessa diferença, não foi enfrentado nenhum problema em relação ao compartilhamento de arquivos via Monticello. O único inconveniente foi a acentuação de palavras, que foi resolvida com código passando a ser escrito sem acentos.

Seaside: O Seaside precisa de algumas configurações iniciais para que o projeto apresentado possa rodar. Segue um conjunto de comandos usado pelo grupo para o desenvolvimento do sistema.

- Para fazer funcionar a acentuação, pare o serviço normal e inicie o *Encoded*. No workspace, basta fazer:

  ```
  WAKom stop.
  WAKomEncoded startOn: 8000.
  ```

  Selecione o texto digitado e, no menu, escolha "`Do it`".

- Abra o Seaside em um navegador apontado para `http://localhost:8000/seaside`. Na seqüência:
  - Vá em Config e digite o login "`admin`" e a senha "`seaside`".
  - Vá em Add Application, digite "`Elefante`" e clique em "`Add`".
    - `Add library: BibliotecaElefante`, `Add`.
    - `Root component: TarefaElefante`, `Save`.
    - `Session class: override`, `SessaoElefante`, `Save`.
- Agora o endereço `http://localhost:8000/seaside/elefante` está ativo e funcionando.

Magma: Magma é um banco de dados orientado a objetos, que é capaz de persistir objetos tanto localmente quanto em um servidor Magma remoto.

Para instalar o Magma, precisa-se executar as seguintes linhas no workspace do ambiente Squeak:

```
Installer squeaksource project: 'Installer'; install: 'Installer-Core'.
Installer universe
answer: 'username' with: 'admin';
answer: 'password' with: 'seaside';
install: 'Magma seasideHelper'; install.
```

Feito isto, pode-se criar o repositório (ou banco de dados), que será usado pela nossa aplicação para armazenar e recuperar objetos. Depois deve-se abrir uma instância do workspace e executar o código abaixo:

```
MagmaRepositoryController create: 'ElefanteBD' root: Dictionary new.
MagmaServerConsole new open: 'ElefanteBD'; processOn: 51001.
```

Após alguns segundos, o Magma criará o repositório.

Ajax: O sistema Elefante utiliza a tecnologia AJAX (*Asynchronous JavaScript and XML*), o que possibilita um ganho em usabilidade do sistema. O *framework* utilizado foi o *Scriptaculous*.

A mais significativa vantagem do uso desta tecnologia é a redução de tráfico de dados, pois com o seu uso o servidor precisa apenas atualizar as partes da página web, cujo conteúdo foi realmente alterado. Com isto, o usuário ganha uma maior usabilidade e um sentimento de que está em uma aplicação flexível (*desktop*), visto que o carregamento de um site web é escondido.

O script mais comumente utilizado é o SUUpdater, que habilita a renderização de apenas algumas partes da página web, sem carregar o conteúdo completo. Este foi o utilizado na aplicação Elefante.

Para o correto funcionamento da tecnologia AJAX na aplicação, é necessário realizar sua configuração. Para isto, seleciona-se a configuração do aplicativo Elefante em `http://localhost:8080/seaside/elefante` e clica-se no botão `Configure`. Em `Add Library`, escolhe-se `SULibrary`. Feito isso selecione o botão `Add`, para adicionar a nova biblioteca na aplicação e o botão `Save`, para salvar as alterações realizadas. Desta maneira realiza-se a associação da biblioteca para o uso das novas funções. Neste, pode-se usar todas as funções do *framework Scriptaculous* no Seaside.

A biblioteca SULibrary contém todos os códigos JavaScript para o framework e é indispensável.

Selenium: *Plugin* para o Firefox que permite realizar testes sobre a interface.

classe SortCriteria: Para o método de ordenação que implementamos funcione, é necessário importar a classe SortCriteria.

Objetos da classe SortCriteria extendem o padrão SortCollection com uma capacidade de ordenação dinâmica, fácil de usar e ainda assim muito poderosa, podendo ordenar múltiplas colunas ou atributos de uma coleção de objetos. É fácil de arranjar a ordenação em modo crescente ou decrescente. É simples porque a classe funciona como as SortedCollection's. É poderosa porque pode-se ordenar qualquer seqüência de atributos em um objeto. É dinâmica porque um objeto da classe SortCriteria e suas colunas podem ser criados "*on the fly*" por uma interface de usuário, possibilitando ao usuário escolher a ordenação que quiser e em ordem crescente ou decrescente.

O arquivo a ser importado está sendo entregue juntamente com o código desenvolvido e chama-se `SortCriteria_Squeak_v1.st`.

2.1 Ferramentas que Já Haviam Sido Usadas

Também foi feito uso intensivo de ferramentas que já haviam sido usadas na fase anterior deste exercício-programa, as quais são listadas abaixo:

Squeak Source: Repositório para compartilhamento de código-fonte.

Monticello: Pacote presente no Squeak. Usado para acessar o site Squeak Source, carregar versões do repositório e fazer *merge* com o código da imagem.

SAR: Os comandos usados no *workspace* para gerar o arquivo `.sar` foram os seguintes:

```
zip := ZipArchive new.
mczStream := RWBinaryOrTextStream on: (String new: 10000).
```

```
workingCopy := MCWorkingCopy forPackage: (MCPackage new name: 'Elefante').
version := workingCopy newVersion.
version fileOutOn: mczStream.
(zip addString: mczStream contents as: 'Elefante-dsp.111142.mcz')
   desiredCompressionLevel: 0.
zip addString: 'self fileInMonticelloZipVersionNamed: '
               'Elefante-dsp.111142.mcz''.' as: 'install/preamble'.
zip writeToFileNamed: 'Elefante.sar'
```

3 Modelo do Sistema

Devido a novas exigências percebidas durante o desenvolvimento da interface web, fizeram-se necessárias algumas alterações nos atributos e no comportamento das classes que haviam sido implementadas na fase anterior do exercício-programa. No entanto, tais mudanças não interferiram no modelo conceitual do sistema, ilustrado na Figura 1, mostrando que ele foi flexível o suficiente para a inclusão das mudanças. O diagrama de classes UML exibe as classes e suas disposições, relações e dependências.

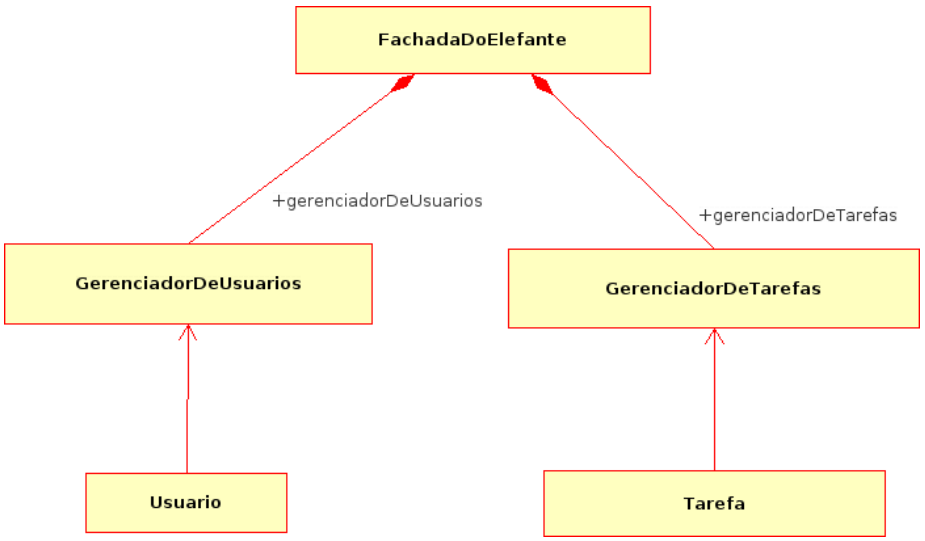

Figura 1: *Diagrama de modelo conceitual.*

No entanto, ao contrário do que ocorreu na outra fase, esta exigiu, com uma maior freqüência, que o grupo marcasse encontros presenciais. O trabalho remoto foi prejudicado pela grande dependência existente entre os módulos que tinham que ser desenvolvidos.

De acordo com o que foi dito no relatório anterior, usou-se o padrão de projeto fachada, possibilitando acessar as demais classes do sistema Elefante. A classe de fachada chama-se `FachadaDoElefante`. O núcleo do sistema continua sendo composto pelas classes `Usuario` e `Tarefa`. As mesmas são acessadas pela fachada por meio de classes que contêm as principais atividades do sistema em seus métodos, chamadas aqui de gerenciadores: `GerenciadorDeUsuarios` e `GerenciadorDeTarefas`.

O modelo conceitual da Figura 1 é estendido e detalhado em um diagrama de classes completo, com atributos e métodos, na Figura 9.

Somando as 5 classes, foram implementados 62 métodos. Todos os métodos passaram nos testes disponíveis para validar a aplicação nesta fase de desenvolvimento, somado aos demais testes desenvolvidos pelos membros do grupo. A classe que possui mais métodos (25) é a `Tarefa`, por ter mais atributos, de modo que possui mais métodos de acesso. A classe que possui mais funcionalidades implementadas em seus métodos (21) é a `FachadaDoElefante`, uma vez que da mesma se opera toda a aplicação.

Seguem outros diagramas UML desenvolvidos.

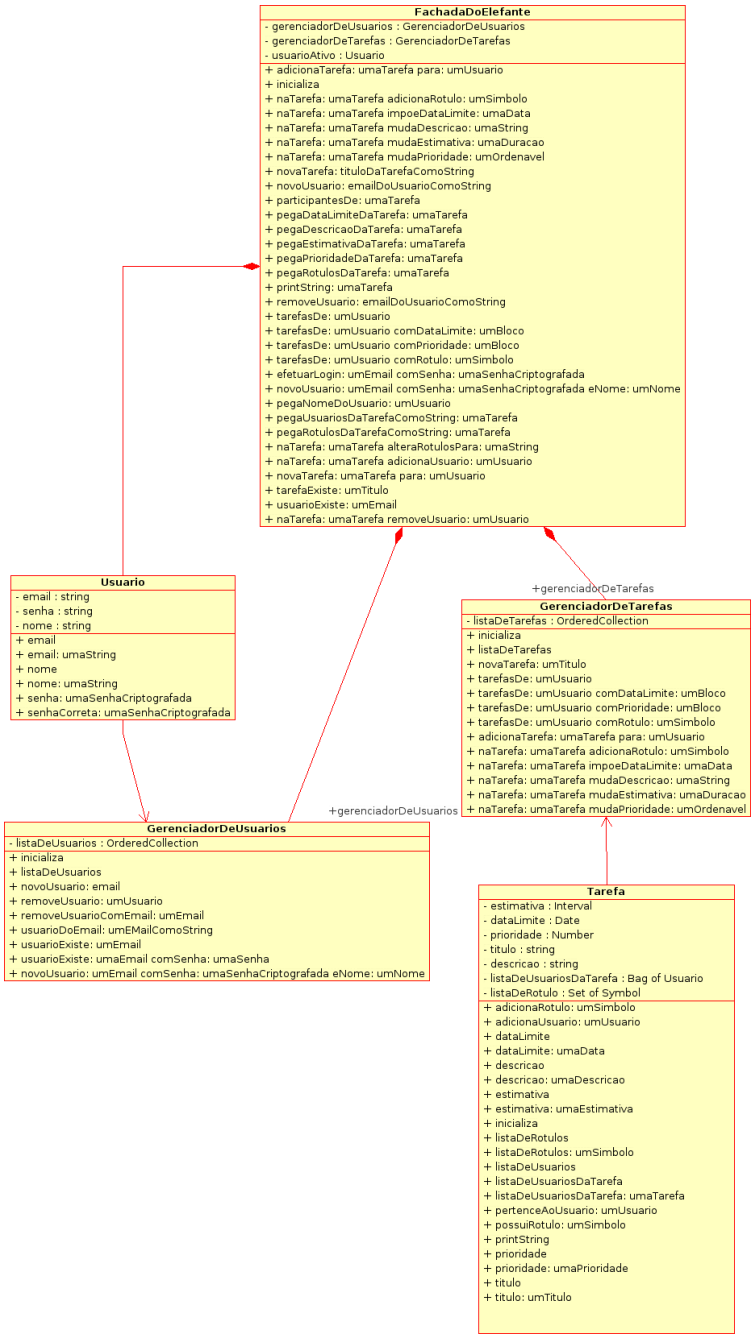

Figura 2: *Diagrama de classes.*

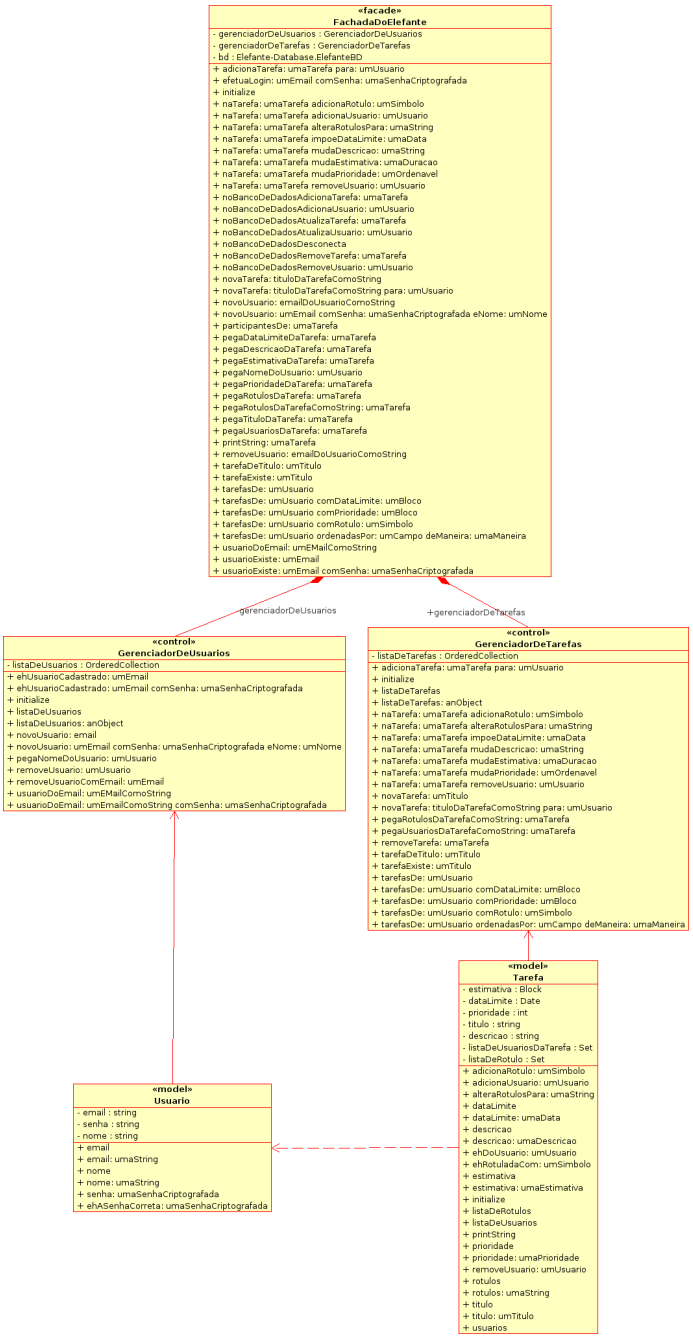

Figura 3: *Diagrama de classes: modelo completo.*

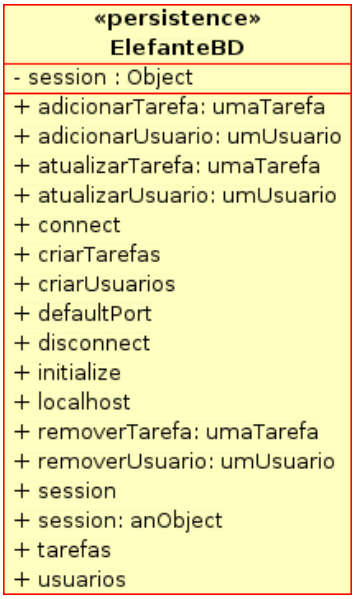

Figura 4: *Banco de dados.*

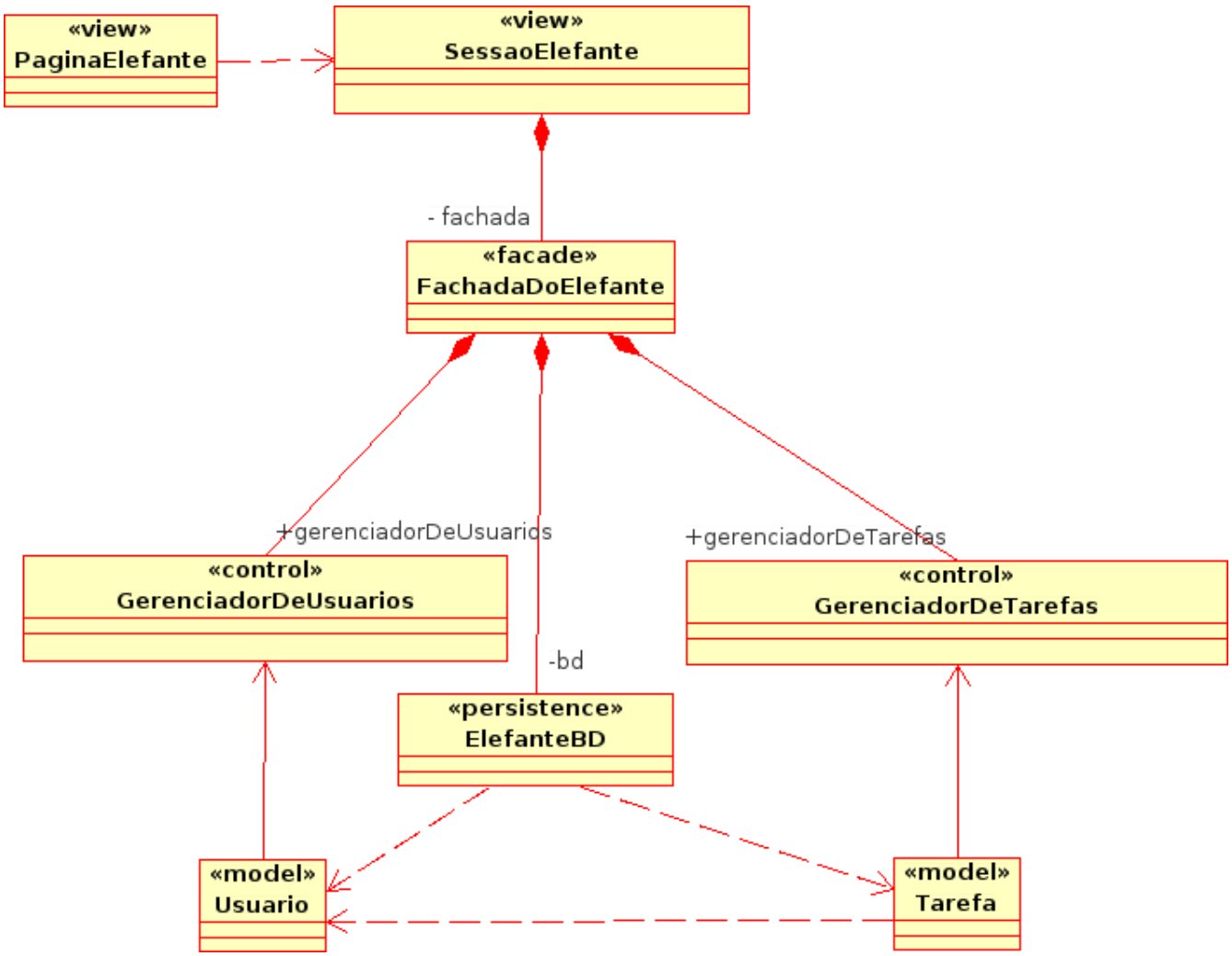

Figura 5: *Modelo do Elefante.*

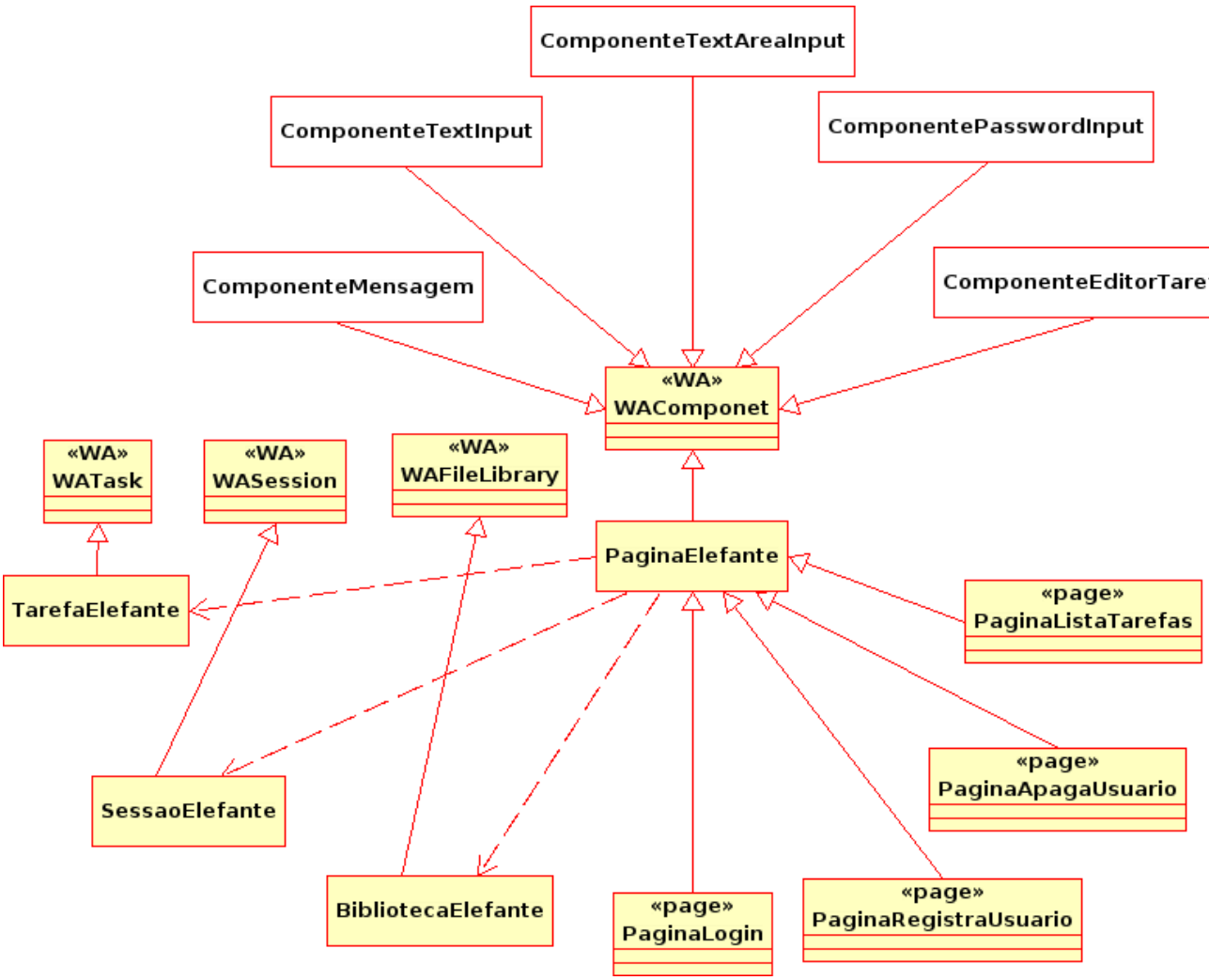

Figura 6: *Seaside.*

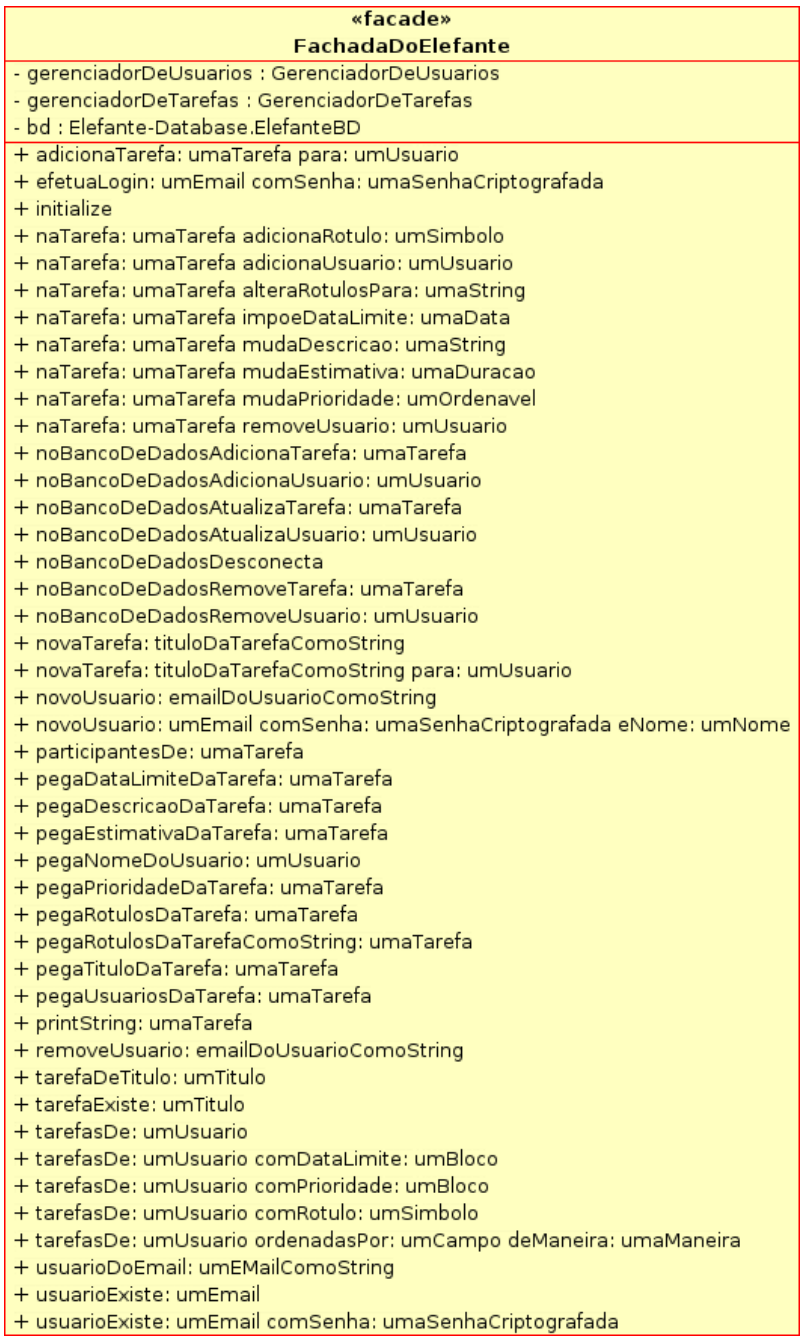

Figura 7: *Fachada.*

Figura 8: *Modelo-controle.*

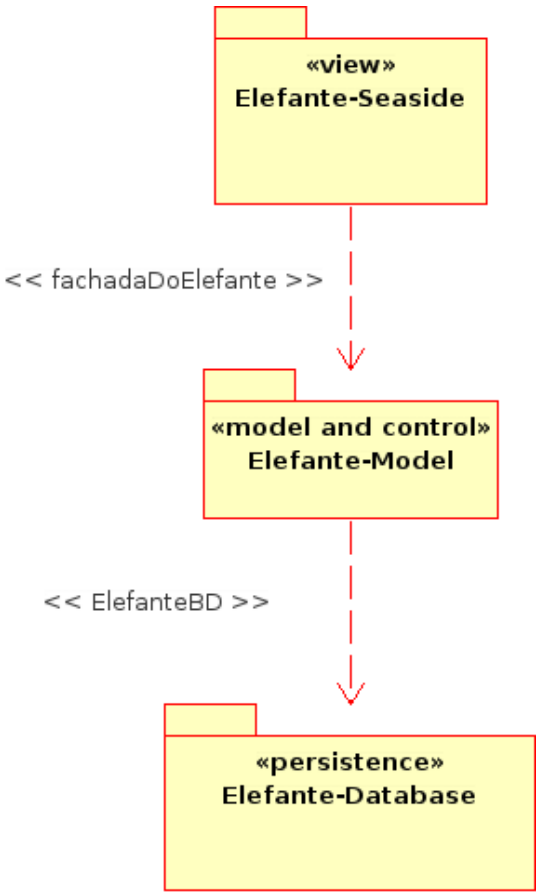

Figura 9: *Pacotes.*

4 Classes Implementadas

Nesta fase, as classes foram separadas em quatro categorias:

Elefante-Model: Contém o sistema básico sobre o qual a interface web é desenvolvida.

Elefante-Seaside: Trata-se da interface web, construída através do uso do *Seaside*.

Elefante-MagmaDatabase: Implementa o sistema de banco de dados para manutenção da persistência dos dados.

Elefante-Tests: É constituída por um conjunto de testes básicos para o modelo.

Cada uma dessas categorias é detalhada nas seções a seguir.

4.1 Categoria `Elefante-Model`

Usuario: É a classe referente ao usuário do sistema, ou seja, aquele que deve ser lembrado das tarefas cadastradas. Contém apenas o atributo `email` e os dois métodos de acesso ao mesmo. Tem como colaborador a classe `GerenciadorDeUsuario`.

Tarefa: Esta classe representa o que uma tarefa possui dentro do sistema. Possui sete atributos: `estimativa`, `dataLimite`, `listaDeRotulos`, `prioridade`, `titulo`, `descricao` e `listaDeUsuarios`. Além dos métodos de acesso, é composta também por mais outros 9 métodos que têm como funções inicializar os atributos, adicionar um usuário a uma tarefa, adicionar um rótulo a uma tarefa, listar os usuários da tarefa, indicar se possui data limite, rótulo e prioridade e imprimir os dados da tarefa. Colabora com a classe `GerenciadorDeTarefas`.

GerenciadorDeUsuarios: É através desta classe que a fachada manipula um usuário. É responsável por adicionar e remover um usuário, verificando se um usuário existe ou não para executar tais tarefas. Contém uma lista de usuários que foram adicionados. Colabora com a classe `Tarefa` e compõe a classe `FachadaDoElefante`.

GerenciadorDeTarefas: Fornece métodos que são utilizados pela fachada para manipulação de tarefas. Além da adição de tarefas à lista de tarefas que é seu atributo, permite diversos acessos às tarefas adicionadas, tais como adição de rótulo, imposição de uma data limite, alteração da descrição, alteração da estimativa de duração e mudança de prioridade. Também são disponibilizados métodos para consulta, que permitem selecionar as tarefas por usuário, data limite, prioridade ou rótulo. Além de colaborar com a classe `Tarefa`, a classe `GerenciadorDeTarefas` também faz parte da composição da classe `FachadaDoElefante`.

4.2 Categoria `Elefante-Seaside`

4.3 Categoria `Elefante-MagmaDatabase`

4.4 Categoria `Elefante-Tests`

Contém testes básicos para partes específicas do modelo. Trata-se de uma subclasse de `TestCase`. Seus métodos fazem uso intensivo da mensagem `assert:`.

5 Padrões das Melhores Práticas em Smalltalk

Foram aplicadas alguma regras descritas no livro Smalltalk: Best Practice Patterns, de Kent Beck.

Devemos confessar que não entendemos todos os padrões nesta primeira leitura. Um exemplo é o padrão Objeto Método (Method Object). O que é aquilo? Não encontramos nenhuma necessidade de aplicação deste padrão em nosso código.

Além da aplicação dos padrões sugeridos no livro, também percebemos alguns que passamos a obedecer. Por exemplo, em busca de métodos menores, de modo a evitar repetição de código, fez-se muito uso de *delegação*.

Antes, para verificarmos se um determinado elemento existia em um conjunto (classe `Set`), fazíamos uso da mensagem `do:`, como no método `Tarefa>>ehRotuladaCom: umSimbolo`:

```
ehRotuladaCom: umSimbolo
   listaDeRotulos do: [:r | (r = umSimbolo) ifTrue: [^ true]].
   ^ false
```

O mesmo serviço pode ser feito com o uso da mensagem `includes:`

```
ehRotuladaCom: umSimbolo
   ^ listaDeRotulos includes: umSimbolo
```

Diversos dos padrões aplicados referem-se mais à modelagem que à codificação, mas mesmo estes últimos possuem um importante impacto na capacidade de comunicação do código a outros programadores.

5.1 Padrões Implementados

Método Composto (*Composed Method*): A divisão do programa em métodos priorizou a delegação de tarefas únicas a serem realizadas por métodos específicos, cada um com poucas linhas de comprimento.

Método Construtor (*Constructor Method*): A criação de instâncias das classes é executada por meio de métodos construtores. Tais métodos localizam-se em categorias (protocolos) de classe denominadas(os) *criação de instância*. A maior parte das classes possui apenas o construtor *new*, não sendo necessária a passagem de parâmetro algum.

Método Construtor com Parâmetro (*Constructor Parameter Method*): Nenhum foi usado até agora. Até agora, a criação de objetos só exigiu a criação de estruturas a serem usadas, tais como OrderedCollection, Bag e Set, as quais podem iniciar desprovidas de conteúdo sem problemas. Criar algum? Se sim, colocar o método no protocolo *privado*.

Método Construtor Atalho (*Shortcut Constructor Method*): Não foi usado nenhum.

Método de Questão (*Query Method*): Usado para testar uma propriedade de um objeto, devolvendo um valor booleano. O prefixo de seu nome é a cadeia "`eh`"—representando "`é`"—, já que evitamos usar acentos no código. Os métodos de questão são colocados em protocolos chamados *teste*. Exemplos: Usuario»ehASenhaCorreta: (antigo senhaCorreta), Tarefa»ehDoUsuario: (antigo pertenceAoUsuario), Tarefa»ehRotuladaCom: (antigo possuiRotulo), GerenciadorDeUsuarios»ehUsuarioCadastrado (antigo usuarioExiste). Ao fazer todas essas substituições, fez-se uso intensivo da mensagem "Senders of" dos métodos, o que facilitava a substituição das chamadas de mensagens em todos os métodos que as continham (colocar isso num parágrafo que só fala das substituições?).

Comentário de Método (*Method Comment*): Este padrão nos fez enxergar que diversos de nossos comentários na versão do sistema entregue na fase 1 eram óbvios demais e poderiam ser inferidos diretamente do código em Smalltalk. Como exemplo, todos os nosso métodos de acesso eram do tipo:

```
UmaClasse>>umAtributo
   "Devolve o valor de umAtributo."
   ^ umAtributo
```

Não obstante, havia comentários que mais confundiam do que documentavam o código. Um exemplo é o método GerenciadorDeTarefas»adicionaTarefa:para:, que continha o seguinte comentário:

> "Associa uma tarefa a um usuário (e vice-versa)."

Esse comentário continha dois erros: ① não é uma tarefa que é associada a um usuário, mas sim um usuário que é associado a uma tarefa, já que é o usuário que entra como parte do Bag listaDeUsuarios de uma tarefa; ② o "vice-versa" não é válido, pois foi decisão tomada no projeto orientado a objetos do sistema que, para obter as tarefas de um usuário seriam varridas todas as tarefas inseridas no sistema em busca do usuário na listaDeUsuarios de cada tarefa.

Esse mesmo padrão foi usado para a anotação, na forma de comentários no código-fonte, de decisões importantes ainda a serem tomadas, os quais, após a eventual resolução do problema, poderiam ser removidos.

Mensagem de Decomposição (*Decomposing Message*): a fim de fazer partes de um cálculo para depois obter o todo, enviamos diversas mensagens a `self`. Como exemplo podemos citar o método GerenciadorDeUsuarios»removeUsuarioComEmail:

```
GerenciadorDeUsuarios>>removeUsuarioComEmail: umEmail
   self removeUsuario: (self usuarioDoEmail: umEmail)
```

Super: Usado em todos os métodos `new` implementados, fazendo com que a criação de objetos obedeça ao padrão de criação do *Smalltalk*.

Modificando o Super (*Modifying Super*): Conforme discussão no fórum, removemos todos os métodos de classe `new`. Além disso, os métodos `initialize` fazem chamada para o `initialize` da superclasse.

Delegação Simples (*Simple Delegation*): O melhor exemplo de aplicação deste padrão no sistema desenvolvido é a `FachadaDoElefante`. Esta classe delega diversas mensagens para as classes `GerenciadorDeUsuarios` e `GerenciadorDeTarefas` sem modificá-las.

Estado Comum (*Common State*): Este padrão foi seguido naturalmente pelos integrantes do grupo uma vez que sua aplicação é direta a partir da modelagem UML, a qual evidencia o estado comum dos objetos de uma classe por meio das variáveis de instância. Para máxima fidelidade ao padrão, todas as variáveis de instância foram declaradas em ordem decrescente de importância na definição da classe.

Estado Variável (*Variable State*): Apesar de o nosso sistema aceitar alguns estados variáveis (e.g., uma tarefa não precisa necessariamente possuir uma descricao ou uma data limite), optamos por não adotar este padrão. Tal decisão levou em conta a relação ganhos/perdas ao se ter que implementar um dicionário de propriedades opcionais às tarefas bem como seus respectivos métodos de acesso. Não nos pareceu valeu a pena tanto trabalho.

Inicialização Explícita (*Explicit Initialization*): Usamos este padrão para aumentar a facilidade de entendimento do código, melhorando sua comunicação. O preço pago por isso é a perda de flexibilidade, já que a cada adição ou remoção de uma variável de instância faz-se necessária uma edição do método de inicialização. Desta forma, somente inicializamos as variáveis de instância que possuem um valor padrão que realmente é usado no sistema. Isto é feito pelos diversos métodos `initialize`, os quais acertam explicitamente os valores. A mensagem de classe `new` é sobrescrita para que o método de inicialização seja chamado na criação de novas instâncias.

A Fazer

- Acertar o select que nos rendeu -.3 na nota do EP1.
- Testar os testes do Hugo no Selenium.
- Testar o acesso concorrente na mesma porta.
- Email para o fórum: Todo o problema descrito surgiu da vontade de programar em português. Após diversas tentativas, percebi que não importa como os métodos "new"e "inicializa"são escritos, sempre haverá chamadas repetidas do "inicializa".

  Depois de ler os padrões Super, Extending Super e Modifying Super (do livro Smalltalk Best Practice Patterns, de Kent Beck), concluí que não dá para tentar traduzir "initialize"para "inicializa". O melhor mesmo é sobrescrever o método "initialize"e, nesta nova implementação, fazer uma chamada para "super initialize". Conforme o XXXXX mencionou, o initialize sempre é chamado pelo método "new", de modo que nem faz mais sentido sobrescrevê-lo como eu fazia antes.

  Kent Beck ainda cita que a melhor solução pode ser usar o padrão "Default Value Method"para representar valores padrões e então sobrescrever apenas esses métodos, mas eu ainda não cheguei nesta página. ;-)
- Incluir o capítulo sobre o padrão fachada novamente?
- Fazer as referências bibliográficas aos novos livros consultados.
- Incluir citacao do site do seaside: http://www.swa.hpi.uni-potsdam.de/seaside/tutorial

* Classe Tarefa: ordem dos atributos: titulo listaDeUsuarios listaDeRotulos dataLimite descricao estimativa prioridade.
* Classe Usuario: email senha nome

Referências

BLACK, A. P., DUCASSE, S., NIERSTRASZ, O. & POLLET, D. (2007). *Squeak by Example*. Switzerland: Institute of Computer Science and Applied Mathematics of the University or Bern.

GUZDIAL, M. (2001). *Squeak: Object-Oriented Design with Multimedia Applications*. Prentice-Hall.

JONATHAN, M. (1994). *Introdução à Programação Orientada a Objeto*. XIV Congresso da Sociedade Brasileira de Computação, XIII Jornada de Atualização em Informática.

KON, F. (2008). Trabalhos de mac 441/5714. `http://www.ime.usp.br/~kon/MAC5714/trabalhos.html`. Último acesso em abril de 2008.

WESSELS, S. (2008). The laser game tutorial. `http://squeak.preeminent.org/tut2007/html/`. Último acesso em março de 2008.

Manifesto de Descontentamento

Gostaríamos de expressar nosso descontentamento com a sede por notas baixas demonstrada pelo monitor. Tal comportamento foi claramente identificado pelas mensagens enviadas ao fórum da disciplina. A impressão que ficou foi de que o fato de diversos grupos da classe terem ido bem nas fases anteriores do exercício-programa foi mal interpretado. Em vez de considerar que todos esforçaram-se por fazer um trabalho bem-feito, cumprindo com os requisitos exigidos no enunciado, foi dada preferência à crença de que foi exigido pouco trabalho. O fato é que, ao menos no nosso grupo, houve muito esforço para entregar um trabalho bem feito, completo e no prazo. Nunca houve a intenção de enganar o monitor ou deixar de fazer algum requisito do EP. Nessa tentativa, integrantes do grupo precisaram desmarcar compromissos, dormir MUITO POUCO e deixar coisas importantes de lado.
Só nos resta este manifesto de descontentamento com esta atitude. Que fique a lembrança de que está-se lidando com adultos e ameaças de rebaixamento de nota são infantis demais para nós.