MAC0441 - Programação Orientada a Objetos
Fase 2

Marcio Masaki Tomiyoshi, 5381892 - masaki@linux.ime.usp.br
Fernando Lemes da Silva, 2371843 - ferlemes@gmail.com
André Jucovsky Bianchi, 3682247 - drebs@linux.ime.usp.br
Ricardo Alexandre Bastos, 5382170 - rakc@terra.com.br

28 de maio de 2008

1 Introdução

Esta fase do projeto consistiu em produzir uma interface web para o `Elefante` e implementar duas funcionalidades extra[1].

Duas questões foram deixadas em abeto na Fase 1. A primeira tratava de como deveriam se relacionar as classes `Tarefas` e `Usuarios`: no modelo que tínhamos imaginado cada instância de `Usuario` teria uma lista de objetos do tipo `Tarefa`, e vice versa. A segunda questão era sobre a função da classe `Calendario` que não estava muito clara para nós.

Durante o desenvolvimento da interface web com o Seaside, percebemos que o modelo tinha sido feito pensando em uma interface diferente da que deveríamos implementar. A estruturação da interface pensando numa aplicação web nos moldes que resultou ajudou-nos a perceber que o ideal seria dar ao `Calendario` a função de um Gerenciador de Tarefas pessoal para cada `Usuario`.

1.1 Seaside e Smalltalk

Esta segunda fase demandou muito mais da perfeita compreensão do código Smalltalk do que as anteriores. As estruturas de fechamentos providas pela linguagem e o esquema de callbacks do Seaside dão bastante flexibilidade para criar o fluxo da aplicação.

Uma parte do modelo do `Elefante` amadureceu. A classe `Lembrador` foi deixada para ser pensada mais pra frente, e `Calendario` foi introduzido.

Além disso, foram criados os componentes da aplicação web: `WAElefante`, `WAFormTarefa`, `WAListaDeTarefas`, `WAListaDeUsuarios`, `WALogin`, `WAMensagens`, `WAMenu`, `WAAjaxMenu` e `WARemoveUsuario`. Acreditamos que o código esteja consistente o suficiente para que os nomes das classes seja auto-explicativo quanto às suas funções.

[1] http://www.ime.usp.br/ kon/MAC5714/trabalhos.html#fase2

2 Testes

Estamos enviando no pacote os arquivos de testes do Selenium atualizados para nosso ambiente. Tivemos que fazer algumas alterações de `clickAndWait` para `click` por causa da funcionalidade extra escolhida de utilizar Ajax para algumas requisições.

Foi introduzido um *pause* de um segundo para evitar que o teste `TesteCompartilhamentoTarefas` realizasse a checagem da quantidade de tarefas antes das tarefas terem sido removidas. Pode ser necessário que essa pausa seja incrementada, mas um segundo foi suficiente durante a realização de nossos testes.

Outra modificação necessária foi a remoção do teste `chooseOkOnNextConfirmationAndWait` que, após a implementação de AJAX travava a execução do teste. Sua remoção nos fez padronizar com o testeCompartilhamentoDeTarefas.

3 Funcionalidades extras

As funcionalidades extra, que começaram a ser implementadas somente depois que todos os testes passaram a funcionar corretamente, foram a persistência usando o banco de dados *Magma*, o uso da ferramenta *Magritte* e implementação de *AJAX*.

3.1 Persistência de usuários e tarefas com o Magma

Para utilizar o Magma, a instância singleton de `GerenciadorDe-Tarefas` mantém uma sessão aberta com um banco de dados Magma local com a arvore de usuários, seus calendários e suas tarefas. Todos os métodos que alteram tarefas e usuários mexem imediatamente no banco de dados local.

Se no futuro for interessante, é fácil modificar o sistema para usar um banco de dados remoto: basta alterar o método `GerenciadorDeUsuarios >> conectaRepositorio` com as opções corretas.

É importante salientar a necessidade do Magma já estar instalado na imagem para a correta execução deste EP, já que este banco de dados não vem por padrão nem na imagem `squeak-web`.

3.2 Magritte

Após o término da implementação do banco de dados, acreditamos que a ferramenta *Magritte* forneceria uma boa funcionalidade ao nosso sistema, possibilitando a criação de formulários e listas mais facilmente, o que, infelizmente, não se concretizou.

Inicialmente, a ferramenta aparentou ser de fácil utilização, já que apenas com a criação de alguns métodos de classe praticamente todo o formulário era gerado automaticamente, podendo detectar os erros desejados.

Porém, logo após a criação do primeiro formulário feito com o *Magritte* não conseguimos encontrar uma opção para utilizar as tags *id* necessárias para a execução correta dos testes. Os campos do formulário já tem uma *id* pré-definida,

que não encontramos meios de modificar. Foram encontrado os métodos *ajaxId:* e *id* em classes de "decoração", que aparentavam ser o que desejávamos alterar, mas não conseguimos utilizar esses métodos, já que pertenciam a classes que não acessávamos diretamente.

Outra dificuldade encontrada foi quando necessário checar se as duas senhas digitadas para cadastro de usuário conferiam, mas acreditamos que conseguiriamos solucionar após melhor entendimento da ferramenta.

Assim, por ter parecido uma ferramenta bastante simples e poderosa, apesar das dificuldades tentamos continuar a sua implementação através de um "Elefante alternativo", em que não seriam realizados os testes (estes seriam executados na versão padrão). Assim, com facilidade foi criado o formulário de criação de tarefas.

Após sua criação, começamos a implementação de listagem de tarefas, que infelizmente gerou muita dificuldade durante a criação de código, o que nos desanimou e nos fez desistir de continuarmos usando a ferramenta. Apesar disso, o código criado continua no sistema como nosso "Elefante alternativo".

Outro aspecto que dificultou em muito o uso da ferramenta foi a pouca documentação disponível. Fora o tutorial disponibilizado no enunciado do EP, encontramos no próprio site do *Magritte* alguns artigos, mas o foco dos mesmos é no desenvolvimento da ferramenta e não em seu uso.

Para finalizar as funcionalidades necessárias para a entrega desta fase, decidimos então implementar *AJAX*.

3.3 Ajax

A implementação do Ajax através da biblioteca *script.aculo.us* não teve segredos. Em geral, a tarefa difícil foi alterar o código para criar updaters e links com *onclick* ao invés de *callbacks*.

É importante ressaltar que a execução correta dos comandos AJAX necessita da biblioteca SULibrary carregada no Seaside.

4 Conclusão

Utilizar o Seaside ajudou a esclarecer e melhorar a estrutura de objetos do nosso sistema. Desta forma, esta Fase representou um avanço em muitos sentidos em relação à fase anterior, desde a melhora na compreensão do código escrito e qualidade do código gerado, até um melhor esclarecimento sobre questões relativas à modelagem do sistema.

A API do Seaside é vasta e um estudo adequado das opções disponíveis é tarefa longa e enriquecedora.