Um Framework MVC para

Aplicações em Java utilizando Swing.

Alessandro Lemser
Curso de Ciência da Computação
Universidade do Vale do Itajaí – Campus São José
São José, SC 88115-100, Brasil
alemser@brturbo.com

Marcello Thiry
Departamento de Ciência da Computação
Universidade do Vale do Itajaí – Campus São José
São José, SC 88115-100, Brasil
thiry@sj.univali.br

Abstract. *This paper describes an desktop application framework to be used with the Java Swing API. The goal of this framework is to promote the decoupling between the Frames and its actions, turning up the modularity of desktop application systems and centralizing the systems resources in one main controller, making issues about security more manageable and controlled with a declarative way. The framework is based on a very know web framework, Jakarta Struts, where the components nomenclature was maintained to gain some similarity between the two frameworks.*

Resumo. *Este artigo descreve um framework para aplicações desktop utilizando a API Swing do Java. O framework visa promover o desacoplamento dos frames e suas ações, aumentando a modularidade dos sistemas desktop e centralização dos recursos do sistema em um único controlador, tornado questões como segurança mais gerenciáveis e controladas de forma declarativa. O framework tem como base o conhecido framework para desenvolvimento de aplicações web, o Jakarta Struts, onde foi buscado manter uma similaridade entre nomenclatura de componentes.*

Palavras-chave: Framework, Swing, MVC, Java.

Tema: Ingeniería de Software.

Workshop: Ingeniería de Software y Base de Datos.

1. Introdução

O sucesso no desenvolvimento de aplicações para a web nos últimos anos e os novos *frameworks* que surgiram para agilizar este desenvolvimento têm contribuído para o crescimento dos sistemas computacionais como um todo. Dentro deste contexto, uma das iniciativas mais bem sucedidas é o Jakarta Struts (http:// jakarta.apache.org/struts). O Struts é um dos *frameworks* mais utilizados para desenvolvimento web e faz parte do projeto Jakarta que é mantido pela Apache Software Foundation (http://apache.org).

Apesar o Struts ter sido desenvolvido para permitir a construção de aplicações web, muitos dos conceitos utilizados em sua especificação podem ser portados para as aplicações *desktop*. Isto se deve ao Struts ter sido desenvolvido com base em padrões de projeto. Além dos padrões de projetos voltados para soluções web, existem padrões de projetos utilizados que são universais, podendo ser aplicados em qualquer linguagem de programação que suporte a orientação a objetos (GAMMA et al., 1996). Um exemplo de padrão de projeto usado na web que pode ser facilmente portado para outros tipos de aplicação é o *FrontController* (ALUR, CRUPI e MALKS, 2002). Além disso, a essência da API Swing é baseada em padrões de projeto, tendo sua base no padrão (MVC – Model, View, Controller).

A partir do *framework* Struts, este artigo apresenta uma solução semelhante que pode ser utilizada para o desenvolvimento de aplicações *desktop*. A proposta é oferecer um padrão de desenvolvimento e permitir que a maior parte deste desenvolvimento seja feito de forma declarativa. A solução trata, por exemplo, de questões de segurança, onde é possível declarar quais grupos de usuários podem acessar determinadas ações do sistema.

A próxima seção apresenta uma breve introdução aos elementos básicos envolvidos no desenvolvimento de aplicações utilizando Swing. As demais seções explicam o funcionamento do *framework* desenvolvido.

2. Elementos comuns de uma aplicação

Uma aplicação que utiliza o conjunto de classes da API Swing, geralmente pode ser composta dos seguintes elementos (HORSTMANN e CORNELL, 2001a), (HORSTMANN e CORNELL, 2001b):

- Um *frame* principal, que estende `javax.swing.JFrame`, o qual possui o papel de ser o container principal.
- Menus que disparam ações no sistema. Estas ações podem estender `javax.swing.AbstractAction`, ou implementar a interface `javax.swing.Action`.
- Outros *frames* para tratar das interfaces com os usuários, os quais podem ser do tipo `javax.swing.JInternalFrame`, caso a aplicação seja MDI.
- Demais ouvintes de eventos que são registrados para serem "avisados" de ações realizadas sobre os componentes que são adicionados aos *frames*.

Em cada um destes elementos são realizadas diferentes implementações. Em uma aplicação de pequeno porte, onde a lógica está contida nas ações do sistema, a camada de apresentação seria formada pelos *frames* (`javax.swing.JInternalFrame`) que seriam responsáveis por validações de dados e por outros controles de interface. As ações (`javax.swing.`

`AbstractAction`) poderiam conter a lógica da aplicação. Um outro conjunto de classes qualquer poderia fazer o papel dos objetos que persistem os dados em algum banco de dados, isto para um modelo de aplicação em três camadas lógicas.

Por outro lado, em uma aplicação de maior porte, a camada de apresentação seria composta pelos *frames* e por suas ações associadas, pela lógica de negócios que poderia estar em um servidor de aplicação acessível através de chamadas remotas, utilizando, por exemplo, EJBs de sessão (*SessionBean*), e pela camada de persistência. Esta última poderia também estar no mesmo servidor de aplicação, utilizando, por exemplo, EJBs de entidade (*EntityBean*).

3. Elementos do framework

Um *framework* é composto basicamente de um conjunto de classes, interfaces que direcionam o desenvolvimento das aplicações em um determinado padrão (FOWLER, 2003). No *framework* desenvolvido, todas as ações do sistema são declaradas em um arquivo XML assim como os *frames*. Todas as chamadas às ações do sistema são centralizadas em um controlador principal, chamado `ActionController`.

Os elementos então citados na seção 2, agora serão substituídos por classes mais especializadas que devem ser herdadas para adquirir a funcionalidade desejada. O *framework* visa abstrair a maneira como são exibidos os *frames*, como são processadas as ações e, se necessário, se uma determinada ação pode ser realizada pelo usuário que a solicitou. As seções a seguir descrevem os elementos básicos do *framework*.

3.1 ApplicationFrame

Trata-se da classe que deve ser estendida pelo *frame* que será o *desktop* da aplicação. Este *frame* deve ser declarado no arquivo de configuração no elemento `application`. Na seção 3.4 são abordados maiores detalhes sobre este arquivo de configuração.

Esta classe fornece alguma funcionalidade básica como a definição de um `JDesktopPane` e implementação do método `windowClosing` da interface `WindowListener`. Futuramente esta classe pode conter mais funcionalidades.

3.2 ActionInternalFrame

`ActionInternalFrame` é o nome da classe que deve ser estendida para se adquirir funcionalidade de um *frame* conhecido pelo *framework*. Estes *frames* estendem a classe `javax.swing.JInternalFrame`.

Um `ActionInternalFrame` possui informações sobre sua `Action`, sobre o *frame* pai (*desktop*) e sobre a sua definição, a qual foi feita no arquivo XML de configuração que será visto na seção 6.

Uma classe que implemente `ActionInternalFrame` pode fornecer métodos acessores (*get*) e modificadores (*set*) para os campos de entrada de dados, esperando que o *framework* instancie e defina os valores de um objeto caso seja solicitado para tal (ver seção 3.4). Porém, ao invés destes métodos retornaram o componente, devem retornar o seu valor contido nestes. Os nomes destes campos devem corresponder aos atributos do objeto. No exemplo de um cliente que tenha o atributo nome, o Quadro 1 ilustra como ficaria o código no *frame*.

```
...
JTextField nome = new JTextField();
...
public String getNome() {
      return nome.getText();
}
public void setNome(String nome) {
      nome.setText( nome );
}
…
```

Quadro 1 – Trecho de código de uma subclasse de ActionInternalFrame**.**

3.3. Action

As ações do *framework* descendem de `javax.swing.AbstractAction`. Para criar uma ação, é necessário criar uma classe que estenda a classe abstrata `oceano.action.Action`. Fazendo isto é necessário fornecer implementação para dois métodos:

- `void start();`
- `void actionPerformed(ActionEvent e, ActionInternalFrame frame).`

O método `start` será chamado se o objeto que originou o evento for um item de menu ou caso a ação for processada diretamente através do método `processAction` do controlador (seção 3.5). Este método é adequado para dar início à ação. Se a mesma possui um frame associado, ela pode exibi-lo neste método. O método `actionPerformed` é chamado quando a ação é notificada por algum evento ocorrido em um componente a qual ela foi registrada como ouvinte.

Uma ação pode ser adicionada diretamente a um item de menu, com isso, o texto do menu será o que estiver contido no elemento *path*, nas definições da `Action`, as quais podem ser vistas na próxima sessão. Como foi dito no parágrafo acima, quando o menu é acionado, o método `start` da `Action` é chamado.

Para que se possa adicionar uma `Action` como ouvinte de eventos de algum componente ou adicioná-la como item de menu, é necessário solicitar ao controlador principal da aplicação (FrontController) para retornar a `Action` baseado em seu *path*. Maiores detalhes sobre o controlador podem ser obtidos na seção 3.5

3.4. Arquivo de configuração

Para que o *framework* conheça as ações e os *frames* do sistema, é necessário declará-los no arquivo `oceano-config.xml`. O Quadro 2 mostra como ficam dispostas estas declarações no arquivo.

Podem existir mais de um arquivo de configuração. Caso seja necessário outro arquivo, este deve ser informado ao pegar uma instância do controlador principal (ver seção 3.5)

O arquivo mostrado no Quadro 2 define basicamente 3 elementos principais: `frame-definitions`, `action-definitions` e `application`. Ainda há outro elemento que não foi exibido neste quadro, que é o elemento que trata da segurança da aplicação. Este elemento será visto na seção 4.

```
<oceano-config>
   <frame-definitions>
      <frame name="clienteFrame"
                  type="teste.ui.ClienteFrame"
                  vo="teste.Cliente"
                  cache-exp="25"/>
   </frame-definitions>

   <action-definitions>
      <action path="action.cad.cli"
                  name="clienteFrame"
                  type="teste.action.ClienteAction"
                  security-roles="adm,fin"
                  security-controlled-by="user"
                  cache-exp="25"/>
   </action-definitions>

   <application type="teste.ui.FramePrincipal"
            resources="ApplicationResources"/>

</oceano-config>
```

Quadro 2. Arquivo oceano-config.xml.

No elemento frame-definitions são declarados os *frames* da aplicação. Cada elemento `frame` define informações de um *frame* em particular. Os seus atributos são listados na tabela 1.

Tabela 1. Atributos do elemento frame.

Atributo	Descrição	Observações
name	Define o nome lógico do frame.	Pelo seu nome é que um frame é associado com uma ação. Este atributo é obrigatório.
type	Define a classe do frame	Este atributo é obrigatório.
vo	Define qual a classe do *ValueObject* (ALUR, CRUPI e MALKS, 2002) representado pelo frame.	Usado para automatizar algumas tarefas no frame. Este atributo é opcional.
cache	Define se o frame será armazenado em cache.	O valor padrão para este atributo é *true*. Este atributo é opcional.
cache-exp	Define quanto tempo (em minutos) o frame permanecerá em cachê.	O valor padrão caso não informado é 30 minutos. Este atributo é opcional.

No elemento action-definitions são declaradas as ações da aplicação. Cada elemento `action` define informações de uma ação em particular. Os seus atributos são listados na tabela 2.

Tabela 2. Atributos do elemento action.

Atributo	Descrição	Observações
path	Define como a ação vai poder ser chamada dentro do sistema.	O valor de *path* também pode ser uma propriedade do arquivo de propriedade informado no elemento `application` deste mesmo arquivo.
name	Define o *frame* associado com esta ação.	Este atributo é opcional.
type	Define a classe da *action*	Este atributo é obrigatório.
security-roles	Informa um ou mais papéis para acessar esta action.	Os papéis são separados por vírgula Este atributo é opcional.
security-controlled-by	Informa se o gerenciamento da segurança será gerenciado pelo *frameowrk* ou pelo usuário.	As opções válidas são: FRAMEWORK ou USER. Este atributo é opcional. O valor padrão é *framework* quando a segurança está habilitada.
cache	Define se o frame será armazenado em cache.	O valor padrão para este atributo é *true*. Este atributo é opcional.
cache-exp	Define quanto tempo (em minutos) o frame permanecerá em cachê.	O valor padrão caso não informado é 30 minutos Este atributo é opcional.

No elemento application é declarado o frame principal da aplicação e o arquivo de recurso caso exista. Para poder informar no atributo *path* de uma *action* como sendo uma propriedade, o atributo *resource* do elemento *application* deve fornecer um o nome de arquivo de propriedades válido.

3.5 Controlador

O `ActionController` é a classe que fornece acesso a todas as ações configuradas no arquivo `oceano-config.xml`. A classe do controlador implementa o padrão *Singleton* e o acesso a instancia que lida com o arquivo de configurações padrão se dá da de acordo com o quadro 3.

```
//para obter a instancia com base no oceano-config.xml padrão
ActionController controller = ActionController.getInstance();

//para obter a instancia com base em um outro arquivo de configuração
ActionController controller =
        ActionController.getInstance( “nome-config.xml” );
```

Quadro 3. Obtendo uma instância de ActionController.

No método *main* de uma classe qualquer a aplicação gráfica pode ser iniciada a partir do controlador através do método `startApplication`.

Uma ação pode ser adicionada a um menu através do método `getAction(path)`, conforme o quadro 4. Este método retorna uma ação com base no seu *path*. Para processar diretamente uma ação, pode-se chamar o método `processAction(path)`. Executar este método fará com que o método `start` da `Action` seja chamado.

```
//Instanciado um menu
JMenu menu = new JMenu(“Novo”);
menu.add( controller.getAction(“cad.cli.action”) );

...

//Instanciando um botão
JButton button = new JButton(“Salvar”);
button.addActionListener(controller.getAction(“cad.cli.action”) );
```

Quadro 4 – Adicionando uma ação a um menu e a um botão.

4. Segurança

A segurança no *framework* é baseada em usuários e papéis (*users-roles*). Para habilitar o gerenciamento de segurança no *framework* deve-se informar os elementos do quadro 5 no arquivo `oceano-config.xml`.

```
<security-constraints>
   <security-enable>true</security-enable>
   <users-roles>
      <loader-type>default</loader-type>
   </users-roles>

</security-constraints>
```

Quadro 5. Declarando restrições de segurança.

De acordo com o Quadro 5 pode-se observar que o elemento `security-constraints` possui dois elementos filhos. O elemento `security-enable` habilita e desabilita a segurança no *framework*. O elemento users-roles espera o nome de uma classe que implemente a interface `oceano.auth.UsersRolesLoader`. Esta interface define apenas um método: `Properties loadUsersRoles()`. O valor *default* indica que será utilizado o `UsersRolesLoader` padrão do *framework*, o qual lê as informações a partir de um arquivo chamado `roles.properties` que deve ficar no *classpath* da aplicação. A classe que implementa a interface é a `oceano.auth.DefaultUsersRolesLoader`.

Um objeto que implemente a interface `UsersRolesLoader` deve fornecer a implementação para o método citado no parágrafo acima e retornar as propriedades informando o nome do usuário (*user*) como chave e seus papéis (*roles*) como valores. Se existir mais de um papel (*role*), devem ser separados por vírgula.

Quando uma aplicação que utiliza o *framework* realiza uma operação de *login*, ela deve informar à classe `oceano.auth.SecurityAssociation` um *subject* (`javax.security.auth.Subject`) e um *principal* (`java.security.Principal`). Para maiores informações a respeito das classes Subject e Principal e sua relação com autenticação e autorização, consulte Java Authentication and Authorization Service (JAAS) em http://java.sun.com/products/jaas/.

5. Conclusão

O *framework* desenvolvido visa reduzir o acoplamento que pode existir entre as ações e os *frames* de uma aplicação *desktop* escrita em Java. Muitos desenvolvedores, devido a pouca experiência ou desconhecimento de boas práticas com a linguagem, acabam permitindo que as telas (*frames*) do sistema assumam um duplo papel, apresentação e controle, fazendo com que os *frames* sejam as próprias ações, ou seja, implementem `Action` ou `ActionListener`. Deixando clara a diferenciação entre estes dois componentes, *frames* e ações, o *framework* promove um desenvolvimento baseado em boas práticas e padrões de projeto.

A centralização das chamadas às ações em um único controlador, facilita o gerenciamento dos recursos utilizados. Também é possível que cada caso de uso tenha seu controlador em particular, tornando mais legível o arquivo de XML de configuração e proporcionando uma maior modularidade e, ainda, tornando os módulos da aplicação mais independentes.

Com relação a questões de segurança, o *framework* oferece uma forma declarativa, centralizada e nos padrões do JAAS (*Java™ Authentication and Authorization Service*).

Um dos próximos passos é a avaliação da utilização do *framework* por programadores acostumados a desenvolver aplicações para a web utilizando Struts. Espera-se que estes tenham uma curva de aprendizado menor para o desenvolvimento de aplicações *desktop*, com relação à criação de telas e ações.

6. Referências

ALUR, D., CRUPI, J. e MALKS, D. **Core J2EE Patterns – As melhores práticas e estratégias de design**. Rio de Janeiro: Ed. Campus, 2002.

HORSTMANN, C. e CORNELL, G. **Core Java 2. Volume I – Fundamentos**. São Paulo: Ed. Makron Books, 2001.

HORSTMANN, C. e CORNELL, G. **Core Java 2. Volume II – Recursos Avançados**. São Paulo: Ed. Makron Books, 2001.

FOWLER, M. **Patterns of Enterprise Application Architecture**. Addison Wesley, 2003.

GAMMA, E. et al. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley, 1996.

JAAS Tutorials. Disponível em http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/tutorials/index.html. Acessado em 20/06/2004.

Java Authentication and Authorization Service (JAAS). Disponível em http://java.sun.com/products/jaas. Acessado em 29/06/2004.