UNIVERSIDADE FEDERAL DA BAHIA

INSTITUTO DE MATEMÁTICA

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

DISCIPLINA: LINGUAGENS PARA APLICAÇÃO COMERCIAL

Beans Binding – JSR 295

Salvador-Ba

Março - 2009

GUSTAVO RAMOS ALMEIDA

Mini-Seminário - Beans Binding – JSR 295

Atividade apresentada como requisito para avaliação no curso de Ciência da Computação da Universidade Federal da Bahia na disciplina Linguagens Para Aplicação Comercial orientada pelo professor Adonai Estrela Medrado.

Salvador-Ba

Março – 2009

SUMÁRIO

1. INTRODUÇÃO

Desenvolvedores desejam cada vez mais ferramentas que acelerem a construção de software. Boas ferramentas devem permitir o reuso de componentes de maneira simples, evitar trabalho repetitivo do programador, gerar código baseado em modelos, etc. Tudo isso para que o desenvolvedor se concentre nas regras de negócio, que é o verdadeiro problema que precisa resolver, e não perca tempo com limitações da linguagem ou da IDE.

A *Java Specification Request* (JSR) 295 [2] visa definir uma API que simplifica a conexão entre um par de Java Beans, para mantê-los em sincronia. Esta conexão será configurável: conversão de tipos e validação podem ser aplicados antes da atualização ser efetuada.

Neste documento serão abordados os principais aspectos do Java Beans Binding, vantagens, desvantagens e uma breve demonstração de como usá-lo.

2. MOTIVAÇÃO

Grande parte do tempo gasto no desenvolvimento de aplicativos com interfaces gráficas (*Graphical User Interface* – GUI) envolve simplesmente transferência de dados do domínio dos objetos para os componentes visuais e vice-versa.

Programadores que desejavam sincronizar dois objetos geralmente criavam seus próprios padrões de sincronia, escreviam várias linhas de código para converter, atualizar e fazer validação nos objetos.

A JSR 295 vai reduzir a quantidade de código tedioso e propenso a erros que um desenvolvedor Java Beans deve escrever. Tornando simples a sincronia entre Java Beans.

3. OUTRAS SOLUÇÕES EM VINCULAÇÃO DE OBJETOS

O *Java Beans Binding* (JSR 295) não foi o primeiro projeto a tentar solucionar o problema. O *JGoodies DataBinding* [7] lançou sua versão 1.1 em 2005, e hoje é uma alternativa estável, utilizada em muitos projetos.

O *JFace Data Binding* [8] é outro projeto, mas que ainda está numa fase inicial. A versão 1.0 foi entregue junto com o Eclipse Europa, contendo ainda muitos bugs.

4. JAVA BEANS BINDING JSR 295

A implementação de referência da Sun está sob a licença *GNU Lesser General Public License* (LGPL), que é considerada um meio termo entre a GPL e licenças mais permissivas, tais como a licença BSD e a licença MIT [10].

Estar sob a licença LGPL significa que qualquer um pode:

- Copiar, distribuir, exibir e executar o software;
- Criar outro software baseado nele e fazer uso comercial do software.

Desde que sejam observadas as seguintes condições:

- Deve dar crédito ao autor original;
- Para cada novo uso ou distribuição, deve deixar claro para terceiros os termos da licença deste software;
- Qualquer uma destas condições pode ser renunciada, desde que seja obtida a permissão do autor;
- O software é distribuído no estado em que se encontra, sem custos, e sem qualquer garantia, implícita ou explicita, por parte do autor.

Por ser implementado totalmente em Java, o framework pode ser utilizado nas plataformas X86, x64, Intel Intanium, SPARC, Java Platform Standard Edition e nos Sistemas Operacionais Windows, Linux, Solaris e Apple OS X.

4.1. DEMONSTRAÇÃO DE USO

O NetBeans 6.5 [1], por padrão, já contém todos os pré-requisitos necessários para começar a utilizar o framework, mas as bibliotecas necessárias podem ser encontradas em [4]. Na IDE é possível fazer a sincronização com apenas alguns cliques, mas neste documento será mostrado como ficaria o código se fosse escrito pelo programador.

A vinculação mais comum, aquela que está demonstrada no APÊNDICE A, é entre um objeto do domínio (um POJO – *Plain Old Java Object*) e um componente visual (neste caso, *JTextField*).

4.2. VANTAGENS

Além de solucionar o problema da sincronização de objetos, usar o *Beans Binding* traz outras vantagens para a aplicação. Como o desenvolvedor utiliza o mesmo código de sincronização diversas vezes, em vez de criar o seu próprio, ele tem menos erros no código.

O popular Modelo de Apresentação [9] diz que as regras de negócio devem estar separadas da GUI. Elas devem estar em camadas diferentes. Pois desta forma é possível testar todas as regras de negócios sem ter que instanciar a interface gráfica. Seguir este modelo é muito mais fácil utilizando *Beans Binding*.

4.3. DESVANTAGENS

Há algumas desvantagens ao utilizar um framework de sincronização de dados. Primeiramente, aplicações são mais difíceis de serem “debugadas” por causa da camada extra que o framework traz. Com isto o fluxo do programa fica confuso. Porém isso se torna mais fácil se o desenvolvedor está familiarizado com o framework.

A refatoração fica inviável utilizando as ferramentas atuais, que não dão suporte a modificações de propriedades em strings. Por exemplo, a string “telefone” é usada para dizer ao framework que a sincronização será feita nos métodos *getTelefone()* e *setTelefone()*. Mas se mudarmos a assinatura do método para *getTelefoneResidencial()* e *setTelefoneResidencial()* seria necessário alterar a string para “telefoneResidencial”. IDEs atuais não capturam estas mudanças em strings.[6].

5. CONCLUSÃO

Para desenvolvedores que irão construir aplicativos com GUI é fortemente recomendado o uso de algum framework que faça a sincronização de beans.

Escrever várias linhas de código, passível de erros, é inviável quando já existe uma solução com um real custo/benefício. Principalmente por que suas desvantagens podem ser supridas em breve com a evolução da IDEs e do próprio *Beans Binding*, que tem aumentado a velocidade da sincronização nas novas versões [3].

O maior problema do *Java Beans Binding* atualmente é a necessidade de adequações que precisam ser feitas nos POJO's (*Plain Old Java Objects*) para estes serem sincronizados [5]. Estas modificações podem ser observadas no APÊNDICE B e APÊNDICE C. Alterar o código desta forma em um sistema que possui uma grande quantidade de POJO's pode se tornar inviável.

No entanto, soluções alternativas apareceram para contornar este problema. O *Java ClassBuilder* [11] promete simplificar a adequação de um POJO à vinculação. Para isto basta adicionar uma anotação *@Bindable* no inicio de cada POJO comum para torná-lo "bindável".

REFERÊNCIAS

[1] NetBeans IDE. Disponível em http://www.netbeans.org/. Acessado em 28 de março de 2009.

[2] JSR 295. Disponível em http://www.jcp.org/en/jsr/detail?id=295. Acessado em 27 de março de 2009.

[3] Beans Binding 1.1.1 Beats 1.0´s Butt, Bigtime. Shannon Hickey. Disponível em http://weblogs.java.net/blog/shan_man/archive/2007/10/beans_binding_1_1.html. Acessado 27 de março de 2009.

[4] BeansBinding Documents & Files. Disponível em https://beansbinding.dev.java.net/servlets/ProjectDocumentList;jsessionid=C90241ECDA5F9CFAFA370EE5A181D6A2. Acessado em 28 de março de 2009.

[5] Beans Binding: A Java Data-Binding Solution with a Serious Problem. Disponível em http://www.devx.com/Java/Article/39964. Acessado em 27 de março de 2009.

[6] Understanding JFace data binding in Eclipse, Part 1: The pros and cons of data binding. Disponível em http://www.ibm.com/developerworks/library/os-ecl-jfacedb1/index.html. Acessado em 27 de março de 2009.

[7] Jgoodies DataBinding Framework. Disponível em https://binding.dev.java.net/. Acessado em 27 de março de 2009.

[8] JFace Data Binding. Disponível em http://wiki.eclipse.org/index.php/JFace_Data_Binding. Acessado em 27 de março de 2009.

[9] Presentation Model. Martin Fowler. Disponível em http://www.martinfowler.com/eaaDev/PresentationModel.html. Acessado em 27 de março de 2009.

[10]. LGPL. Disponível em http://pt.wikipedia.org/wiki/LGPL. Acessado em 27 de março de 2009.

[11] Java ClassBuilder. Disponível em http://code.google.com/p/javabuilders/wiki/ClassBuilder. Acessado em 27 de março de 2009.

APÊNDICES

APÊNDICE A – Trecho de código exemplificando uso de Beans Binding

```
...

import org.jdesktop.beansbinding.Bindings;

public class ExampleJFrame extends javax.swing.JFrame {

    public ExampleJFrame() {

        initComponents();

        org.jdesktop.beansbinding.BindingGroup bindingGroup = new
org.jdesktop.beansbinding.BindingGroup();

        org.jdesktop.beansbinding.Binding binding;

        binding =
Bindings.createAutoBinding(org.jdesktop.beansbinding.AutoBinding.UpdateStrateg
y.READ_WRITE, dvd1, org.jdesktop.beansbinding.ELProperty.create("${nome}"),
jTextFieldNome, org.jdesktop.beansbinding.BeanProperty.create("text"),
"binding_nome");

        bindingGroup.addBinding(binding);

        bindingGroup.bind();

    }

private DVD dvd1;

...
```

APÊNDICE B – Código de um POJO comum

```
public class Person  {

   private String firstName;
   private String lastName;

   public String getFirstName() {
      return firstName;
   }
   public void setFirstName(String firstName) {
      this.firstName = firstName;
   }

   public String getLastName() {
      return lastName;
   }

   public void setLastName(String lastName) {
      this.lastName = lastName;
   }
}
```

APÊNDICE C – Código de um POJO capaz de ser sincronizado

```
import java.beans.PropertyChangeSupport;
import java.beans.PropertyChangeListener;

public class Person  {

   private String firstName;
   private String lastName;

   private PropertyChangeSupport support = new PropertyChangeSupport(this);

   public void addPropertyChangeListener(PropertyChangeListener listener ) {
      this.support.addPropertyChangeListener( listener );
   }

   public void removePropertyChangeListener(PropertyChangeListener listener )
{
      this.support.removePropertyChangeListener( listener );
   }

   public String getFirstName() {
      return firstName;
   }

   public void setFirstName(String firstName) {
      String old = this.firstName;
      this.firstName = firstName;
      support.firePropertyChange("firstName",old,firstName);
   }

   public String getLastName() {
      return lastName;
   }

   public void setLastName(String lastName) {
      String old = this.lastName;
      this.lastName = lastName;
      support.firePropertyChange("lastName",old,lastName);
   }
}
```