Injection and conversations, three years with Seam

2009-10-02/StephaneEpardaud@www.lunatech.fr
Injection and conversations, three years with Seam

# @In Speaker stephaneEpardaud;

# @Scope(CONVERSATION)

> Uses

> Abuses

# @In/@Out: bijection

> Do not throw yourself in the tar pit

# Conclusion
Senior Java developer at Lunatech Research


Java EE since 2001

Seam since 2007

Footnote:

I expect Pete Muir to focus on the overwhelming good sides of the few drawbacks I present
Injection and conversations, three years with Seam

We should have a conversation
Let’s have a conversation

# From the Seam manual:

> “A conversation is a unit of work from the point of view of the user. It might span several interactions with the user, several requests, and several database transactions.”

# Features:

> New scope

> (Potentially) longer than a page

> Shorter than a session
When to have conversations

Stateful beans are conversation-scoped by default

Demarcation for user actions

Allows the user to have several conversations at once
When to use conversations?
When to use conversations?

**AWC-1 Spanish wine selection**
Order created 01-Jan-2001 00:00, updated 01-Jan-2001 00:00 (change history)

**Reference:**

**Creator:** Wayne, The Averys Wine Cellars

**Buyer:** Boston Tea Party

**Carrier:** Carl, Carl's Trucks

**Seller:** Wayne, The Averys Wine Cellars

**Carrier reference:** 3684

**Tags** red [x] wine [x]

- New tag: Enter one or more tags separated by spaces
- Popular: controlled priority-1

---

**Order lines (3)**  **Comments (2)**  **Status history**  **Attachments (0)**  **Consignments (0)**  **Sub orders (1)**

**Attachment:**

**Description:**

New Attachment

**Save** or **Cancel**
Mapping conversation to code

@Stateful
@Name("tagActionBean")
public class TagActionBean
    implements TagAction{

    @Begin
    public void addTag(){};

    @End
    public void save(){}

    @End
    public void cancel(){}
    }

@Stateful
@Name("commentActionBean")
public class CommentActionBean
    implements CommentAction{

    @Begin
    public void addComment(){};

    @End
    public void save(){}

    @End
    public void cancel(){}
    }
Mapping conversation to code

@Stateful
@Name("tagActionBean")
public class TagActionBean implements TagAction{
    @Begin
    public void addTag(){};
    @End
    public void save(){}
    @End
    public void cancel(){}
}

@Stateful
@Name("commentActionBean")
public class CommentActionBean implements CommentAction{
    @Begin
    public void addComment(){};
    @End
    public void save(){}
    @End
    public void cancel(){}
}
Why that doesn’t work

# Cannot start a conversation when one is already active

> Use nested conversations?

# How to know which action uses which nested conversation?

> Need to store and forward the conversation IDs

# In the end, we dropped conversations...
Seam is the new Perl?

# Where to begin/end conversations?

> @Begin/@End on methods

> <begin-conversation/> per view

# One of our refactoring was figuring out which is best and stick to it

# We settled for XML

> Different pages use the same code in different contexts
Injection and conversations, three years with Seam

You should get that injection looked at
Bijection you say?

# Bijection is injection and outjection of dependencies

# Way to avoid boilerplate code to get to data

> Provider/Consumer model

> Weak dependencies

# Nice UEL integration

> Useful for JSF
Bootstrapping the injections

Before you can @In[ject] you need to @Out[ject] it

@Stateful
@Name("orderActionBean")
public class OrderActionBean implements OrderAction{

    @Out
    private List<Order> orders;

    @Factory("orders")
    public void initOrders(){
        orders = orderDAO.getOrders();
    }
}
Bootstrapping the injections

# What about order?

@Stateful
@Name("orderActionBean")
public class OrderActionBean implements OrderAction{

    @Out
    private List<Order> orders;

    @Out
    private Order selectedOrder;

    @Factory("orders")
    public void initOrders(){
        orders = orderDAO.getOrders();
    }

    @Factory("selectedOrder")
    public void initOrder(){
        selectedOrder = orders.get(0);
    }
}

Depends on the order of use in the views!!
Bootstrapping the injections

# What about order?

```java
@Stateful
@Name("orderActionBean")
public class OrderActionBean implements OrderAction{

    @Out
    private List<Order> orders;

    @Out
    private int ordersCount;

    @Factory("orders")
    public void initOrders()
    {
        orders = orderDAO.getOrders();
        ordersCount = orders.size();
    }

}
```
Bootstrapping the injections

# What about null?

```java
@Stateful
@Name("orderActionBean")
public class OrderActionBean implements OrderAction{

    @Out
    private Order order;

    @Factory("order")
    public void initOrders(){
        if(...)
            order = orderDAO.getOrders();
    }
}
```

Will be called for each use until non-null
Seam is the new Perl?

# Which style?

```java
@Stateful
@Name("orderActionBean")
public class OrderActionBean implements OrderAction{

@Out
private List<Order> orders;

@Factory("orders")
public void initOrders(){
    orders = orderDAO.getOrders();
}

@Factory("orders")
public void initOrders(){
    if(orders == null)
        orders = orderDAO.getOrders();
}

@Factory("orders")
public List<Order> getOrders(){
    return orderDAO.getOrders();
}
```
Recipes for failure (1)

# Dangers of @Out[jection]

@Stateful
@Name("userActionBean")

public class UserActionBean implements UserAction{

    @Out
    private User currentUser;

    public void login(){
        currentUser = ...;
    }

    public void doSomethingElse(){
        // we forgot to inject currentUser
        // so it will be outjected as null
    }
}
Recipes for failure (2)

# Beware of the contexts used

```xml
<m:message xmlns="http://www.w3.org/1999/xhtml"
            xmlns:m="http://jboss.com/products/seam/mail">
  <m:from name="#{currentUser.name}" address="#{currentUser.address}" />
  <m:to name="#{user.name}" address="#{user.address}" />
  <m:subject>Try this out</m:subject>
  <m:body>Seam sends emails</m:body>
</m:message>
```

@Stateful
@Name("mailActionBean")
public class MailActionBean implements MailAction{

  @In(create=true)
  private Renderer renderer;

  public void send(Order order) {
    Contexts.getConversationContext().set("currentUser", order.getCreator());
    Contexts.getConversationContext().set("user", order.getBuyer());
    renderer.render("/mail.xhtml");
  }
}
Conclusion
Conclusion

# Seam is great, we only focused on problematic cases

# Conversations and bijection are very useful

> Like all new paradigms they require training

> Likely improved in JCDI / Web Beans