OrchestraDialogPageFlowDesign1

This page contains notes from the original design discussion on this topic between Manfred, Thomas, Mario and Simon in March 2008.

This is not a finished design, just a starting point.

defining the flow

The flow should be able to be configured in the faces-config.xml. This should be the main difference to other dialog implementations as this avoids you to learn any new xml configuration. To make this work we need to define conventions.

The default implementation will treat any view below "flow/" as a page associated to a flow named by the path-token following "flow".

e.g. /flow/booking/enterBookingDetails.xhtml belongs to the page-flow named "booking". As soon as the system hits an url outside the "booking" flow the flow will be terminated.

Notice: This implementation should be able to be replaced by a user-defined class so that any other strategy can be chosen.

This example shows the transition from the "main" flow (reviewHotel) (the "main" flow might be optional, any page outside a flow will belong to the "main" flow) to the "booking" flow. The "transition page" (reviewHotel) also defines (booking.success, booking.cancel) what should happen when the booking flow ends gracefully (e.g. using an "ok" or "cancel" button).

```
<navigation-rule>
    <from-view-id>/flow/main/searchHotel</from-view-id>
    <navigation-case>
        <from-outcome>search</from-outcome>
        <to-view-id>/main/reviewHotel.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>

<navigation-rule>
    <from-view-id>/flow/main/reviewHotel</from-view-id>
    <navigation-case>
        <from-outcome>book</from-outcome>
        <to-view-id>/flow/booking/enterBookingDetails.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>booking.success</from-outcome>
        <to-view-id>/flow/main/bookingOk.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>booking.cancel</from-outcome>
        <to-view-id>/flow/main/bookingCancelled.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>

<navigation-rule>
    <from-view-id>/flow/booking/enterBookingDetails.xhtml</from-view-id>
    <navigation-case>
        <from-outcome>review</from-outcome>
        <to-view-id>/flow/booking/reviewBooking.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
```

talking with the flow

Normally a flow will be started by a navigation rule. The above example will start a new flow when reviewHotel returns an action outcome of "book". As always, the navigation will be issued by a commandLink or commandButton.

Since it depends on the action how the dialog needs to be configured, a tag which configures the dialog must be nested. This should allow to use the same flow with different input parameters and return methods.

So, This is the place where we could be able to "talk" nicely with the flow.

An example could be:

```
<h:commandButton action="book">
    <orch:dialogListener listener="#{reviewHotelBean.bookingDialogEvent}">
        <orch:param name="hotelType" value="#{reviewHotelBean.hotelType}" />
        <orch:param name="hotelId" value="#{reviewHotelBean.hotelId}" />
        <orch:outcome dialogOutcome="ok" actualOutcome="booking.success" state="ok"/>
        <orch:outcome dialogOutcome="cancel" actualOutcome="booking.cancel" state="cancel" />
    </orch:dialogListener>
</h:commandButton>
```

The listener is a method with the following signature:

```
public void bookingDialogEvent(DialogEvent event)
```

DialogEvent will provide the state parameter configured by orch:outcome (default=dialogOutcome) and a return object which has been setup by the final action method of the dialog.

orch:param defines the parameters passed to the dialog context. One should be able to inject these values to the actual dialog backing bean

orch:outcome defines the end-states of the dialog.

When such an end stated has been reached, the custom Navigation handler will lookup the caller view id and replaces the outcome with the actualOutcome. Effectively this means that the navigation "booking.success" or "booking.cancel" will be issued. The dialog listener method will be called too.

configuring the dialog backing bean

```
<bean name="bookingDetailsBean" class="my.backing.bean.BookingDetails">
    <parameter name="hotelType" value="#{dialogParam.hotelType}" />
    <parameter name="hotelId" value="#{dialogParam.hotelId}" />
</bean>
```

The dialogParam will be a special Spring setup which allows to get in touch with the parameters passed in by the dialogListener.