DialogManagerFeature

Dialog Manager Feature

Introduction

The Dialog Manager feature of Shale shows promise in helping application developers organize large scale applications into "conversations" with the user, maintaining local state solely for the duration of that conversation (instead of letting it accumulate in session scope until the user logs off). Unfortunately, the current implementation is the subject of several outstanding bug reports:

- SHALE-10 IFrame does not work properly inside Shale dialog
- SHALE-48 Serious issue with dialog state
- SHALE-61 Maintaining dialog synchronization when browser navigation buttons are used

Plus a slew of enhancement requests:

- SHALE-113 Create "Dialog Aware" button panel component
- SHALE-123 Enhancement to Shale Dialogs
- SHALE-153 Allow dialog substate to return information to calling state
- SHALE-175 Dialog statemachine and errorconditions should be queryable
- SHALE-184 Provide new "Dialog Scope" for managed bean scope
- SHALE-188 Add a default outcome to the dialog manager

To address these issues, it seems likely that some rearchitecture will be required. In that case, it also makes sense to go back and articulate the set of requirements that the revised functionality should support, to ensure that they all get met satisfactorily.

Requirements

For an appropriate prioritization, the requirements below are separated into MANDATORY, DESIREABLE, and NICE TO HAVE buckets. Within each bucket, requirements are numbered - so any new requirements added to a bucket later should go to the end. Original requirements that we decide to remove from the list should be visually labelled in some way, to maintain the identity of the requirement numbers.

Mandatory Requirements

1. Make implementation decisions that minimize the amount of new learning for a developer that is already familiar with JSF.
   - Also, leverage existing JSF concepts such as managed beans and programmatic expression evaluation where it is useful.
2. Support modelling of a dialog as a UML state diagram, with states embodying various kinds of processing activity and transitions between states being driven by logical outcomes from these activities.
3. Support configuration of zero or more uniquely named dialogs within the scope of a single web application.
   - An XML based configuration mechanism is required, with support for one or more configuration resources, a default resource name if none is specified, and automatic recognition of configuration resources in the META-INF section of a JAR file.
4. Support the concept of an *action* state that represents a call to an arbitrary method of some arbitrary bean, which returns a logical outcome.
   - Support configuration of the method to be called with a JSF method binding expression
   - Logical outcome returned by this method is used to select the appropriate transition
5. Support the concept of a *view* state that represents the rendering of a particular JSF view, following by a subsequent postback, up to and including Invoke Application phase.
   - Logical outcome returned by the invoked action method is used to select the appropriate transition
6. Support the concept of an *exit* state that causes the current dialog to be completed (throwing away any stored state information).
   - Requires some mechanism for an exit state to provide a logical outcome, in order to drive transitions in a parent dialog
7. Support the concept of a *subdialog* state that allows reuse of a different dialog definition as a "black box" subroutine.
   - Logical outcome returned by the exit state of the invoked subdialog is used to select the appropriate transition.
8. Context data for a currently executing dialog must be maintained while a dialog is in progress, and must be automatically thrown away when the dialog is exited.
   - Must be accessible programmatically in JSF event handlers
9. Support for multiple active dialog instances within a single page.
   - Includes support for multiple instances of the same dialog name.
10. Support for multiple active dialog instances across multiple frames or windows, including popup windows.
    - Includes support for multiple instances of the same dialog name.
11. State synchronization must deal appropriately with use of browser navigation buttons.
12. Implementation must not modify standard JSF semantics like "return null from an action method means redisplay the current view".
13. Must be possible to enter a dialog as a result of a JSF navigation rule.
14. It must be possible to use the dialog management features without relying on any state save/restore facilities of the dialog functionality, as long as the application is managing its own state in this scenario. (Credit for this idea to Paul Spencer <paulsp AT apache.org>.)
15. The framework must provide a configurable mechanism to deal with application exceptions that does not disable dialog functionality. (Credit for this idea to Paul Spencer <paulsp AT apache.org>.)

Desireable Requirements

1. Context information for a currently active dialog instance is accessible via a custom VariableResolver so that it can be referenced with value binding expressions. (May be difficult to disambiguate a single variable name if there are multiple active dialog instances going on.)
2. Use of a particular JSF view in a dialog should be transparent to the page author (i.e. do not require explicit artifacts to be included in the component tree).

3. Use of a particular JSF backing bean (such as a Shale ViewController) in a dialog should be transparent to the backing bean author, unless programmatic access to the state information for the current dialog instance is invoked.
    - If EL expressions can be used to access the state, this can still be fairly transparent.
4. Support of either a custom dialog-defined JavaBean with properties for individual state data, or a generic Map, as the holder of state data.
5. Programmatic entry into, and exit from, a dialog instance (in addition to navigation based entry and exit state based exit).
6. A stable public API to access the static configuration of a dialog (as opposed to the dynamic state and context of an executing dialog instance).
7. The framework should recognize updated configuration files and reload them, without requiring an application restart. (Credit for this idea to Paul Spencer <paulsp AT apache.org>.)

Nice To Have Requirements

1. Be synergistic with any future implementations of fine grained security in Shale.

Prior Art

Shale is certainly not the first framework to attempt an implementation of this concept (although we are trying for a distinctly "JSF feeling" implementation).

- Spring WebFlow
- JBoss Seam
- Jakarta Commons SCXML (State Chart XML)
- Apache Trinidad Dialog Framework

Implementation Approaches

Add a section below for each implementation approach to be considered, including pro/con and tradeoffs in achieving the goals described above.