Keep on Swinging

Productivity layers on top of SWT

Karsten Schmidt
SAP AG
k.schmidt@sap.com
Keep on Swinging - Agenda

- Background
- SAP NetWeaver Developer Studio
- The UI (Container) Framework
- The SAP Viewer architecture
- Dialogs
- Reusable UI components
- Image reuse
- Summary
Keep on Swinging - Agenda

- Background
  - SAP NetWeaver Developer Studio
  - The UI (Container) Framework
  - The SAP Viewer architecture
- Dialogs
- Reusable UI components
- Image reuse
- Summary
Background

SAP NetWeaver Developer Studio

- Beginning of 2002: SAP decides to build an IDE for its Java technology stack based on Eclipse
- The project is staffed with Java developers with mostly a Swing background
- Tool development will eventually be done in several departments (even companies) and several geographic locations
  - Rising number of tool developers with strongly varying backgrounds
- In other contexts – SAP Guis for Windows, Java Environment, HTML – customizable L&Fs were requested and implemented
Background

Goals
- High productivity of tool development
- Consistent UIs across the product
  - “There is only one way to do a specific task”
- Be prepared for different L&Fs

Obstacles
- SWT is a low-level UI approach for Swing programmers
  - Thin abstraction layer of Win32 API
- Basic Swing concepts are missing in SWT
  - Reuse of UI components
  - Reparenting of UI components
  - Support for different L&Fs
- Differing approaches to other concepts
  - Model View separation: Data Models vs. Content Providers
- At that time little SWT documentation and expertise available
Background – Possible approaches

Integrate Swing UIs
- No consistent UI / L&F
- Technically not reliable

Create a Productivity Layer on top of JFace / SWT
- Carry some Swing concepts to the SWT world
- Pave the way for Swing developers
- Hide some of the SWT complexity
- Allow reuse of Swing Data Models
- Pragmatic approach instead of principal discussions

Caution
- Emphasis lies on support for tool development
- Framework functionality not for its own sake
  - Functionality built on demand
  - Completeness not a goal

This is just one possible approach!
Keep on Swinging - Agenda

- Background
- SAP NetWeaver Developer Studio
  - The UI (Container) Framework
  - The SAP Viewer architecture
- Dialogs
- Reusable UI components
- Image reuse
- Summary
SAP NetWeaver Developer Studio – Overview

- Support for standard technologies
  - J2SE, J2EE, Web Services, XML, ...
- Support for SAP technologies
  - Web Dynpro, Data Dictionary, ...
- Additional Framework functionality required
- UI productivity layer is part of Service Layer
Keep on Swinging - Agenda

- Background
- SAP NetWeaver Developer Studio
- The UI (Container) Framework
  - UI containers
  - Subcomponents
  - Layout Data
  - Look & Feel
  - Style Bits
  - “Native” SWT access
- The SAP Viewer architecture
- Dialogs
- Reusable UI components
- Image reuse
- Summary
Rough Architectural Overview

- **SAP UI productivity layer**
  - Adds frequently needed but missing features to JFace/SWT
  - Simplifies development of UIs
  - Supports uniform look and feel of UIs
  - No replacement: native widget access always possible!

- **JFace**
  - UI toolkit on top of SWT (extension without hiding)
  - For common UI programming tasks (e.g. actions and viewers)

- **SWT**
  - Common OS-independent API
  - Tightly integrated with the OS (esp. the Win API)
One Simple Example ...

The same logon popup created by using the different programming models.
UI containers

The SAP UI layer defines a set of UI containers (parent panes)
- Grid layout pane – subcomponents arranged in a grid like manner
- Sash pane – subcomponents are separated by sashes
- Tab pane – each subcomponent is displayed under a tab
- Toolbar pane – specific to build vertical/horizontal toolbars
- Simple pane – provides just a simple composite (full SWT functionality)

Top-level pane is created by a factory on the SWT parent control
- shell
- editor area

Subelements are added by a set of convenience methods
- Simple UI elements (e.g. buttons)
- Complex UI elements (e.g. viewers)
- Other (sub-) containers
UI containers – Examples

- Grid pane
- Tab pane
- Sash pane
Subcomponents in Swing are created and added to their parents

```java
JButton cancelButton = new JButton("Cancel");
pane.add(cancelButton, buttonLayoutData);
```

In SWT, subcomponents are created on their parents

```java
Button cancelButton = new Button(container, SWT.PUSH);
cancelButton.setText("Cancel");
cancelButton.setLayoutData(buttonData);
```

With the SAP UI layer, subcomponents are added to parent panes by using a set of convenience methods with meaningful parameters

```java
IGridLayoutPane glp = PaneFactory.createGridLayoutPane(shell, 3);
Button cancelButton = glp.addPushButton("Cancel", buttonLayoutData);
```
Subcomponents - “Semantic” API Example

Adding a push button to the parent pane (with default layout information)

```java
public Button addPushButton(String text);
public Button addPushButton(Image image);
public Button addPushButton(String text, String toolTip);
public Button addPushButton(Image image, String toolTip);
```

Exactly the same methods exist again for specifying additional layout information

```java
public Button addPushButton(String text, IGridLayoutData layoutData);
public Button addPushButton(Image image, IGridLayoutData layoutData);
...
```

Similar methods exist to add checkboxes, radiobuttons, images, viewers, sub components (panes), ... and even separators.
Layout data – Swing/SWT

In Swing as well as in SWT layout data is set/specified by directly accessing the appropriate public members of the layout object

- Very extensive, flexible, because each layout type (e.g. grid, flow, row, ...) is treated in a similar fashion
- Not very common from an object oriented point of view (data hiding)
- Can be error prone or confusing, because global constants have to be assigned to member variables

Example

```
labelLayoutData.fill = GridBagConstraints.NORTHEAST;
labelLayoutData.anchor = GridBagConstraints.BOTH;
```

- Makes no sense according to documentation
- Leads to IllegalArgumentException at run time
- BUT: possible without even a warning at design time
Concrete layout information can only be specified for subcomponents of grid layout panes

- Reduced flexibility, but sufficient for almost all use cases

Layout data is specified by layout data objects

- Always the last parameter of an “add” method
- Created by factory methods on the grid layout (semantic API)
- Reusable (unlike in Swing)

```java
IGridLayoutData checkBoxLayoutData = glp.createGridLayoutData(IGridLayoutAlignment.CENTER, IGridLayoutAlignment.FILL);
IGridLayoutData buttonLayoutData = glp.createGridLayoutData(true, false);

glp.addCheckBox("Remember my password", true, checkBoxLayoutData);
Button okButton = glp.addPushButton("OK", buttonLayoutData);
```
Look & Feel – Swing

```java
UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());

Java Cross Platform Look & Feel

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

Windows Look & Feel
```
Look & Feel – SAP UI layer

IGridLayoutPane glp = PaneFactory.createGridLayoutPane(shell, 3);

Widget creation delegated to the default widget factory (standard SWT widgets)

IWIDGETFactory widgetFac = new WhiteWidgetFactory();
IGridLayoutPane glp = PaneFactory.createGridLayoutPane(shell, 3, widgetFac);

Widget creation delegated to a specialized widget factory
Style bits - SWT

Widget type is determined by SWT style bit during creation

Button checkBox = new Button(container, SWT.CHECK); // checkbox
Button cancelButton = new Button(container, SWT.PUSH); // push button
...
Label realLabel = new Label(container, SWT.NULL);
Label separator = new Label(container, SWT.SEPARATOR | SWT.HORIZONTAL);

Style bits can be combined although sometimes it makes no sense

Button strange = new Button(container, SWT.CHECK | SWT.RADIO); // ???

No predefined default style bits

Text pwdField1 = new Text(container, SWT.BORDER); // looks good!
Text pwdField2 = new Text(container, SWT.NULL); // unattractive!
// a checkbox with given label and default value ‘checked’
glp.addCheckBox("Remember my password", true, checkBoxLayoutData);

// a simple push button with the given text
glp.addPushButton("OK", buttonLayoutData);

// simple text label
glp.addTextLabel("Password");

// horizontal separator
glp.addHorizontalSeparator();

SWT style bits are computed when creating the widgets based on reasonable defaults and the method parameters
- e.g. viewers are scrollable (SWT.H_SCROLL || SWT.V_SCROLL)
- e.g. text fields are bordered (SWT.BORDER)
Native SWT access

Focus lay on support of tool development, not on the framework aspect

Functionality was added to the SAP UI layer on demand

No promise of completeness

Direct SWT access covers remaining functionality
Native SWT access

“Add” methods return the “native” SWT widget

The widget can then be processed further

Therefore, functionality not provided by the SAP UI layer API can be accessed by processing the native SWT widgets

Example: convert a text field into a password field:

```java
IGridLayoutPane glp = PaneFactory.createGridLayoutPane(shell, 2);
glp.addTextLabel("Password");

org.eclipse.swt.widgets.Text pwdField =
glp.addTextField(false, textFieldLayoutData);

pwdField.setEchoChar('*');
pwdField.setText("default");
...
```
Native SWT access - Simple panes

The complete SWT functionality can be accessed by using a “Simple Pane” as sub component

- Simple Panes are newly created by the pane factory or by adding them to the parent component by add-methods
- They contain one public method which simply returns the pane composite for further processing

Example: using a simple pane to display a picture:

```java
ISimplePane picturePane = glp.addSimplePane(simplePaneLayoutData);
Composite pictureComposite = picturePane.getComposite();

// Canvas widget used to show the image
Canvas imageCanvas = new Canvas(pictureComposite, SWT.NULL);
imageCanvas.setBackground(WHITE);
imageCanvas.addPaintListener(new PaintListener() { ... });
...```

© SAP AG 2002, Keep on swinging, Karsten Schmidt 25
Keep on Swinging - Agenda

- Background
- SAP NetWeaver Developer Studio
- The UI (Container) Framework
- The SAP Viewer architecture
- Dialogs
- Reusable UI components
- Image reuse
- Summary
The SAP Viewer Architecture

Allows to combine
- a standard java (i.e. Swing) table or tree model
- and
- a JFace table or tree viewer

Advantages
- Reuse of existing models
- Viewer reacts (and updates) on model changes
- MVC behavior known from Swing: Model is a well-defined entity
- Swing model classes / interfaces are well documented
- Helpful if both Swing and SWT are targeted
- Configurable by replacing the default label provider (in order to have an application specific decoration of the elements)
- Allows to concentrate on the Data Model
The SAP Viewer Architecture – Example

The following piece of code demonstrates how to use Swing tree model together with a JFace tree viewer

```java
TreeModel defTreeModel;

// Build up a initial set of tree nodes
MutableTreeNode root = new DefaultMutableTreeNode("Root", true);
MutableTreeNode node1 = new DefaultMutableTreeNode("Node 1", true);
MutableTreeNode node11 = new DefaultMutableTreeNode("Node 1.1", true);
MutableTreeNode node2 = new DefaultMutableTreeNode("Node 2", true);

// Create default tree model
defTreeModel = new DefaultTreeModel(root);

// Build up the tree from the created nodes
defTreeModel.insertNodeInto(node1, root, 0);
defTreeModel.insertNodeInto(node11, node1, 0);
defTreeModel.insertNodeInto(node2, root, 1);

// Add SAP tree viewer
treeViewer = gridPane.addSapTreeViewer(defTreeModel, viewerLayoutData);
```
Keep on Swinging - Agenda

- Background
- SAP NetWeaver Developer Studio
- The UI (Container) Framework
- The SAP Viewer architecture

Dialogs
- Reusable UI components
- Image reuse

Summary
Dialogs - Standard

User Dialogs
- Provide static method for easily doing standard user interactions
- Simple delegation to the JFace dialogs

Message Dialog
- Extends JFace message dialog and gives opportunity to show detail information (e.g. stack traces)

```java
MessageDialog.showError("Log Item Viewer", "Details about the error", "...");
```
Dialogs - Specific

Object Selection Dialog
- User can select from a plain list
- Choice is done by (un-)checking the objects

Structured Object Selection Dialog
- Similar, but objects are represented in a structured (tree like) manner
- Sophisticated possibilities to specify tree logic
  - Node checking
    - Handling of folders
    - Preselection
    - ...
  - Handling of grayed nodes
Keep on Swinging - Agenda

- Background
- SAP NetWeaver Developer Studio
- The UI (Container) Framework
- The SAP Viewer architecture
- Dialogs
- **Reusable UI components**
- Image reuse
- Summary
Reusable UI Components

Providing a library of reusable (complex) UI components
- increases the productivity of tool development
- leads to more consistent UIs across the product

Two variants

- Generic UI components:
  - Identify “Usage Patterns”
    - Example: Eclipse property view
  - Find the best UI approach to support that pattern
  - Create a generic UI component implementing the UI approach
  - “There is only one way to do a specific task”

- Specific UI components
  - Example: Method Editor
Generic UI Components – Tree Selector I

Enterprise Beans

- session.beans
  - Stat1Bean
    - env-entry
      - EnvEntryXY
    - ejb-ref
      - ejb/Cmp1Bean
      - ejb-local-ref
    - security-role-ref
      - SomeSecRole
    - security-identity
    - resource-ref
    - resource-env-ref

- entity.beans
  - Cmp1Bean
    - cmp-field
    - env-entry
    - ejb-ref
    - ejb-local-ref
    - security-role-ref
    - security-identity
Generic UI Components – Tree Selector I

Tree for Selection

Add / Remove Buttons
Specific editor for selection
Please fill in a role link or role name corresponding to an existing security role.
Specific UI Components – Java Method Component

Specific UI component for creation / editing of java methods
- Used several times within the J2EE toolset (usage pattern)
- Inner components (e.g. modifiers) are again reusable UI components
Keep on Swinging - Agenda

- Background
- SAP NetWeaver Developer Studio
- The UI (Container) Framework
- The SAP Viewer architecture
- Dialogs
- Reusable UI components
- Image reuse
- Summary
Image reuse

Goals
- Only one image for a specific purpose (object type, action)
- Install the minimal number of copies of any given image
- Make the (re)use of existing images as convenient as possible
- Make the Eclipse Workbench images available

Background
- In SAP R/3, a huge library of images was centrally maintained
- In the early days of SAP NetWeaver Developer Studio development we found around 20 copies of saplogo.gif in the plugins folder
- At the same time, different images were used for the same purpose

J2EE Perspective
Development Configurations Perspective
Image reuse

Three levels of image reuse
- Eclipse Workbench images
- SAP standard images
- Images from another plugin

Images or image descriptors are accessed via static methods
- All images are stored in an Image Registry and are therefore finally disposed

```java
// images defined in the eclipse workbench
SapIdeUtilImages.getWorkbenchImage(ISapSharedImages.IMG_OBJ_FOLDER);

// SAP standard images (defined as constants)
SapIdeUtilImages.getSapImage(SapImage.ICON_REFERENCE_LIST);

// images stored as image file in any plugin (addressed by descriptor or id)
SapIdeUtilImages.getImageFromIconsDirectory("com.tssap.editors","folder/img.gif");
```
Keep on Swinging - Agenda

- Background
- SAP NetWeaver Developer Studio
- The UI (Container) Framework
- The SAP Viewer architecture
- Dialogs
- Reusable UI components
- Image reuse
- Summary
Summary

Goals of the SAP UI productivity layer

- Increase the productivity of tool development
- Support consistent UIs across SAP NetWeaver Developer Studio
- Bridge the gap for Swing experienced developers
- Allow and support reuse of
  - Complex (existing) Swing tree or table models
  - Generic or specific UI components
  - Images

The SAP UI productivity layer is fully based on SWT

It is just one possible (pragmatic) approach – not yet another contribution to the principal SWT / Swing discussion

Further information on SAP NetWeaver

- SAP Developer Network (http://sdn.sap.com)