Nicolas SEBBAN
2 octobre 2003

SWT: The Standard Widget Toolkit

www.eclipse.org

Plan

- Principes de fonctionnement
  - Présentation de SWT
  - Stratégie
  - Gestion des ressources
- Principes d'utilisation avec JFace
  - Présentation de JFace
  - Les Viewers
  - Pré-requis

Présentation de SWT

- Bibliothèque graphique IBM OpenSource
- Utilisée dans Eclipse
- Procure des widgets natifs d'une manière indépendante de l'OS

Présentation de SWT

- Bibliothèques graphiques multi-plateformes difficile à écrire/maintenir car système de widgets complexes et différences entre plateformes → SWT résout ce problème: techniques d'implémentation de bas niveau

- SWT implémentée sur différentes plateformes: utilisation combinaison JAVA et JNI spécifique à chaque plateforme

Stratégie de SWT

- SWT: 100% Java !
- JNI utilisé pour invoquer l'OS
- Mapping « one-to-one » entre méthodes natives Java et appels à l'OS
- Stratégie appliquée strictement!!

Stratégie de SWT

Exemple: sélection dans zone de texte

Composant Text

```
/* Select positions 2 to 5 */
    text.setText ("0123456780");
    text.setSelection (2, 5);
```

Windows

```
ublic void setSelection (int start, int end) {
      OS.SendMessage (handle, OS.EM_SETSEL,
tart, end);
  }
```

```
lass OS {
      public static final int EM_SETSEL = 0xB1;
      public static final native int SendMessage (int
Wnd, int Msg, int wParam, int lParam);
      ...
  }
```

```
 NIEXPORT jint JNICALL
ava_org_eclipse_swt_internal_win32_OS_SendMess
ge__IIII
    (JNIEnv *env, jclass that, jint hWnd, jint Msg, jint
Param, jint lParam)
  {
      return (jint) SendMessage((HWND)hWnd, Msg,
Param, lParam);
  }
```

Motif

```
 public void setSelection (int start, int end) {
       int xDisplay = OS.XtDisplay (handle);
       if (xDisplay == 0) return;
       OS.XmTextSetSelection (handle, start, end,
OS.XtLastTimestampProcessed (xDisplay));
       OS.XmTextSetInsertionPosition (handle, end);
    }
```

```
 class OS {
       public static final native void XmTextSetSelection
(int widget, int first, int last, int time);
       public static final native int
XtLastTimestampProcessed (int display);
       public static final native void
XmTextSetInsertionPosition (int widget, int position);
       public static final native int XtDisplay (int widget);
       ...
    }
```

```
 JNIEXPORT void JNICALL
Java_org_eclipse_swt_internal_motif_OS_XmTextSetS
lection
     (JNIEnv *env, jclass that, jint widget, jint first, jint
last, jint time)
    {
       XmTextSetSelection((Widget)widget, first, last,
time);
    }
```

Stratégie de SWT

→ implémentation du composant Text non portable MAIS L'APPLICATION QUI L'UTILISE L'EST!

```
/* Select positions 2 to 5 */
    text.setText ("0123456780");
    text.setSelection (2, 5);
```

Classe Text différente pour chaque plateforme mais signature de chaque méthode publique est la même

Stratégie de SWT

Pourquoi ne pas faire comme ceci ?

```
public void setSelection (int start, int end) {
        nativeSetSelection (start, end)
    }

static final native void nativeSetSelection (int start, int end);

JNIEXPORT void JNICALL Java_org_eclipse_swt_widgets_text_nativeSetSelection
       (JNIEnv *env, jclass that, jobject widget, jint first, jint last)
    {
    #ifdef WINDOWS
        HWND hWnd = SWTGetHandleFromJavaWidget (widget);
        SendMessage(hWnd, Msg, wParam, lParam);
    #endif
    #ifdef MOTIF
        Widget *w = SWTGetHandleFromJavaWidget (widget);
        Display xDisplay = XtDisplay (w);
        if (xDisplay == NULL) return;
        XmTextSetSelection (w, start, end, XtLastTimestampProcessed (xDisplay));
        XmTextSetInsertionPosition (w, end);
    #endif
    }
```

Stratégie de SWT

- Séparation responsabilités: développeurs windows/motif…
- Mieux vaut tout écrire en java: langage de haute niveau, classes réutilisables, …
- Tout est au même endroit
  - pratique pour debugger
  - performance tuning (une fois dans le natif, les perfs ne dépendent que de l'OS)
- Etc…

Gestion des ressources

- SWT utilise ressources de l'OS → nécessité de libération
- 2 règles à retenir:
  - Si on le créé, on le libère
    - Ressources allouées automatiquement dans constructeurs des widgets SWT
    - Libération par méthode dispose()

`Font f=control.getFont()` → dispose()? NON

Gestion des ressources

- La libération d'un parent libère les enfants
  - Un widget ne peut pas exister dans l'OS sans parent → si parent libéré, enfants libérés
  - Attention, certains composants comme Font, Color ne sont pas des enfants
  - Un widget SWT ne libèrera jamais une ressources qu'on a allouée (sinon règle 1 brisée)

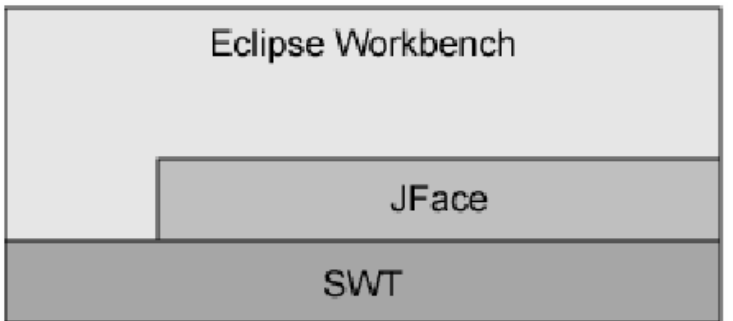
Gestion des ressources

- Il faut libérer! Ne pas attendre que l'OS libère les ressources en fermant l'appli. Risque de manque de ressource pour l'application et pour autres programmes.

- Outil « Sleak »: aide recherche ressources non libérées

Présentation de JFace

- SWT procure widgets natifs « bruts »: Button, Label, Text, List, Tree, Table, …
  - → pas pratique à manipuler (String, …)
- JFace: surcouche « outil » de SWT
  - Simplifie développements en SWT
  - Travaille AVEC SWT, SANS le cacher.



Présentation de JFace

- JFace regroupe modèles d'utilisation de SWT
- Permet par exemple manipulation objets métier, plutot que String, …

Les Viewers

- Viewers: utiliser widgets avec objets métier
- En SWT: convertion des objets métier en String, Images, … gérées par widgets natifs
- Viewers: adaptateurs sur widgets SWT
- Viewers pour widgets SWT non-triviaux: List, Tree, Table, …

Les Viewers

- Chaque Viewer associé à 1 widget SWT
- Viewers utilisent 3 notions principales:
  - Input
  - ContentProvider
  - LabelProvider

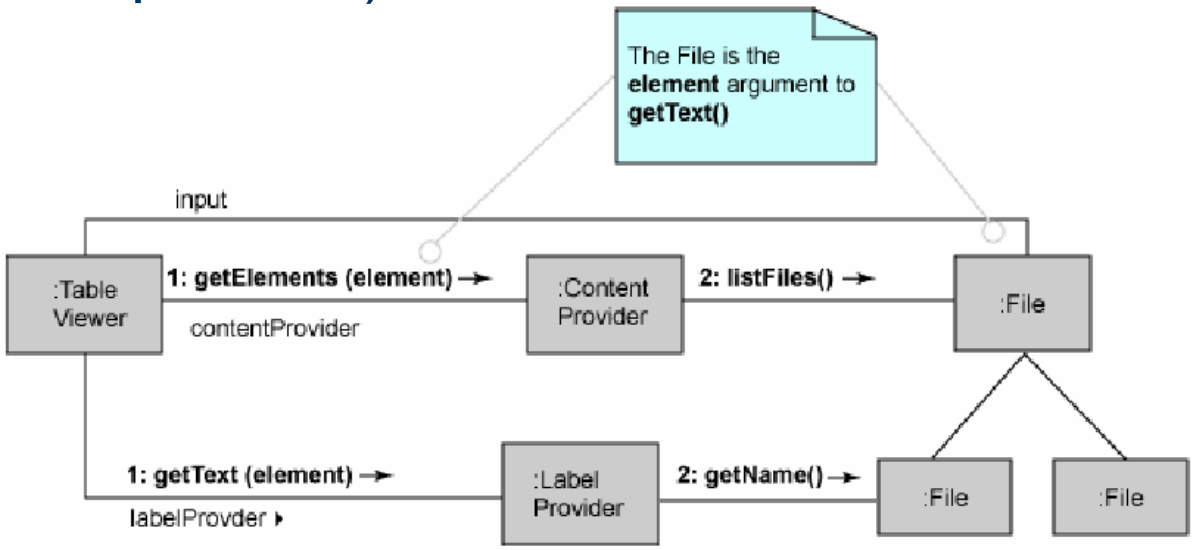
Les Viewers

- Input:
  - Objet principal que le viewer affiche/édite
  - N'importe quel objet, simple ou composé
- ContentProvider:
  - Interface, définit protocole pour obtenir informations sur le contenu de l'objet Input
  - Informé du changement d'Input
  - Des ContentProvider spécialisés implémentés pour différents types de Viewers
    - Ex: StructuredContentProvider: procure une liste d'objets pour un Input donné. Utilisé par viewers de type « liste »

Les Viewers

- LabelProvider
  - Interface, produit éléments graphiques spécifiques au contenu d'un viewer (obtenu par le content provider)

Ex:



Autres atouts de JFace

- Viewers de type « liste » offrent possibilités de tri et de filtres personnalisés (ViewerSorter, ViewerFilter)
- TableViewer peut gérer édition des cellules
- Widgets/outils réutilisables
  - WIZARD!!!
  - Widgets d'eclipse…

Pré-requis pour SWT/JFace

- Dans le class path:
  - `C:\eclipse-2.1.0\plugins\org.eclipse.core.boot\boot.jar`
  - `C:\eclipse-2.1.0\plugins\org.eclipse.jface_2.1.0\jface.jar`
  - `C:\eclipse-2.1.0\plugins\org.eclipse.runtime_2.1.0\runtime.jar`
  - `C:\eclipse-`

Conclusion

- Stratégie de SWT efficace: performant, multiplateforme
- JFace indispensable, principe viewers efficace
- Widgets existants réutilisables
- Autres avantages/aspects: plugins eclipse, support ActiveX, …

Références

- www.eclipse.org:
  - Articles: SWT: The Standard Widget Toolkit
    - Part 1: Implementation Strategy for Java Natives
    - Part 2: Managing Operating System Ressources
  - Guide Viewers (aide d'eclipse)
- www-106.ibm.com/developerworks:
  - Article: Using the Eclipse GUI outside Workbench
    - Part 1: Using JFace in stand-alone mode