Mise en place d'un SashForm dans une section, de façon similaire au pattern Master/DetailsPage

par David CHAUTARD (http://david-chautard.developpez.com/)

Dernière mise à jour :

Dans ce billet nous allons présenter une solution permettant d'intégrer un SashFom dans une section, de façon similaire au Pattern Master/Details page.

I - Présentation

Nous développons une application Eclipse RCP dans laquelle nous utilisons le pattern Master/Details page, avec les **UI Forms**. Cela nous permet d'afficher nos données sous forme d'arbres et de pages détaillées, contenant les propriétés de l'élément édité. Ce pattern permet de faire un scrolling entre la Master page et la Details page, grâce à l'utilisation d'un **sashForm**.

Figure 1 : Exemple de Master/Details page

Dans les Details page (ie : la partie droite de la figure précédente), nous avons différentes sections qui permettent d'afficher nos données. Parmi ces sections, nous nous intéressons à celles composées d'un arbre ou d'une liste, et de l'ensemble de données de l'élément sélectionné dans l'arbre, qui permettent de les éditer. Il s'agit du même principe que le pattern Master/Details page énoncé ci-dessus, sans la notion de scrolling entre les deux parties.

Figure 2 : Exemple de section avec arbre

II - Le besoin

La problèmatique avec ce type de section est que dans certains car il est difficile de visualiser à la fois le contenu de l'arbre, et les attributs de sa selection, ce qui ne facilite pas l'édition des champs de données.

Figure 3 : Exemple section avec arbre et attributs tronqués

La solution serait donc de mettre en place un composant particulier Section, intégrant un sashForm, afin de pouvoir réduire la taille de l'arbre et d'agrandir les champs des attributs, pour faciliter leur saisie.

Figure 4 : Exemple section avec sashForm affichant les attributs en entier

Mais également de réduire la tailles des champs des attributs pour visualiser la contenu de l'arbre.

Figure 5 : Exemple section avec sashForm affichant l'arbre en entier

Nous souhaitons donc créer ce composant qui accueillera un sashForm.

III - La solution mise en place

Pour résoudre ce problème, la solution mise en place est la suivante :

Tout d'abord, nous créons la classe abstraite *ADetailsComposite*, qui étend la classe *SashForm* et non plus Composite. Le fait d'étendre cette classe permet de gérer automatiquement les rafraichissements des composants lorque le sash est déplacé. Cette classe abstraite permet de gérer la création de l'arbre, ainsi que le composant contenant les champs des attributs.

Cette classe contient une méthode de classe et deux méthodes abstraites.

- la méthode *createContent* est appelée dans le constructeur.
  Cette méthode créé la partie de gauche de la section, qui contient l'arbre. Celle-ci, tout comme la partie droite, qui affiche les attributs, sont contenus dans un composant ScrolledComposite, afin de faire apparaitre des scrollbarres si nécessaire, lorsque le sashForm est déplacé. Ces deux composants sont des variables de la classe.

Ensuite nous initialisons la répartition de ces composant dans la section :

```
protected int[] sizeDetailsComposite = new int[]{55, 45};
```

Elle sera mise à jour dans la classe fille, lors de la création de la partie droite, et appliquée via la méthode *setWeight(int[] weights)* de la classe mère *SashForm*.

Pour la mise à jour de la partie droite lorsqu'on clique sur un élément de l'arbre, nous avons un listener sur l'arbre, qui appellera la méthode *createDetails*, avec la sélection en paramètre.

```
myTreeViewer.getTreeViewer().addSelectionChangedListener(new ISelectionChangedListener() {
public void selectionChanged(SelectionChangedEvent event) {
 if (event.getSelection() instanceof IStructuredSelection) {
  IStructuredSelection lSelection = (IStructuredSelection) event.getSelection();

  selection = (EObject)lSelection.getFirstElement();

  if(selection != null)
   createDetails(selection);
 }
});
```

- la méthode *initialize* qui gère les labelProvider et contentProvider de l'arbre. Cette méthode sera implémentée dans la classe fille.
- la méthode *createDetails* qui gère la contruction du composant de droite. Cette méthode sera implémentée dans la classe fille.

```
protected void createDetails(EObject selection) {

  if(getWeights() != null && getWeights().length == 2){
   sizeDetailsComposite = getWeights();
  }

  GridData gridData1 = new GridData();
  gridData1.grabExcessHorizontalSpace = true;
  gridData1.horizontalAlignment = GridData.FILL;
  gridData1.verticalAlignment = GridData.FILL;
  gridData1.grabExcessVerticalSpace = true;

  //create composite
  if (tableComposite !=null && compositeRight != null){
   tableComposite.dispose();
   compositeRight.dispose();
  }

  compositeRight = new ScrolledComposite(this, SWT.NONE | SWT.H_SCROLL | SWT.V_SCROLL);
  compositeRight.setBackground(getDisplay().getSystemColor(SWT.COLOR_WHITE));
  compositeRight.setExpandHorizontal(true);
  compositeRight.setExpandVertical(true);
  compositeRight.setMinSize(200, 100);

  /* debut construction du contenu*/
  ...
  tableComposite
  ...
  /* fin construction du contenu*/

  compositeRight.setContent(tableComposite);
  Point size = monContenu.computeSize(SWT.DEFAULT, 100);
  compositeRight.setMinSize(size);

  setWeights(sizeDetailsComposite);
```

```
    mform.getToolkit().paintBordersFor(tableComposite);
    tableComposite.setLayoutData(gridData1);

    layout();

  }
```

Nous avons donc un composant abstrait qui gère la partie gauche de notre section. La partie droite qui est spécifique, sera implémentée dans la classe fille.

IV - Remerciements

Je voudrais remercier ... pour sa relecture, ainsi que tous ceux qui ont pu m'aider à la rédaction de cet article.