

# Construire un composant "glisser - déposer" avec Google Web Toolkit

2<sup>e</sup> partie

*Nous avons vu dans la première partie de l'article, comment intégrer des scripts JavaScript au sein d'un module Google Web Toolkit (GWT) et comment introduire du code JavaScript au sein des classes Java à l'aide de JSNI.*

Cela nous a permis de construire un composant panel accueillant des widgets glissantes et des widgets cibles : les widgets glissantes devant être déposées sur des widgets cible<sup>2</sup> compatibles pour que l'opération de « glisser-déposer » soit acceptée par notre composant.

Dans cette seconde partie, nous allons enrichir notre composant. Cela nous permettra de montrer comment rajouter un mécanisme événementiel à un composant GWT. Nous verrons aussi des fonctionnalités avancées de JSNI, comme l'appel de méthodes Java à partir du code JavaScript.

## La seconde version de notre composant

Notre composant n'est pas encore très utile, car si les widgets se déplacent, le reste de l'application n'en sait rien. Pour pallier ce manque, nous allons créer un système d'événements. Nous sommes là dans le JAVA très classique, quoiqu'un peu simplifié : on ne définit pas de classe événement dans le GWT, les informations contenues traditionnellement dans l'objet événement sont passées directement comme paramètre à la méthode réflexive. Notez que rien ne nous empêche de passer par le truchement d'un objet événement. Les API du GWT n'en utilisent pas.

Pour cela, nous allons créer une interface qui doit être implémentée par tous les objets qui vont se mettre à l'écoute des événements de « glisser-déposer » émis par notre panel :

```
public interface DragDropListener {

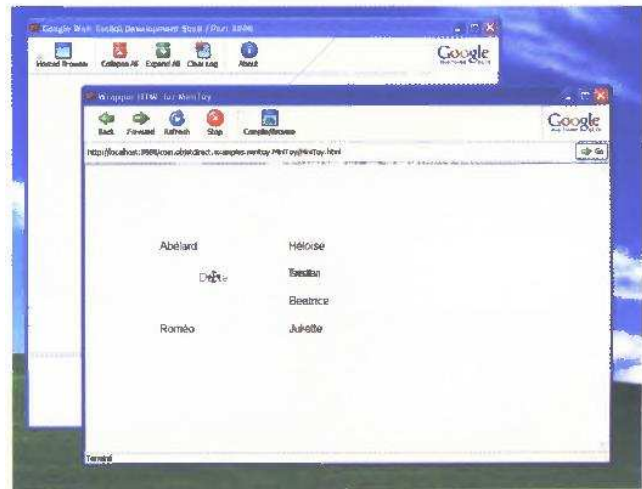
    public void onDrop(Widget draggable, Widget target);
}
```

Il faut maintenant inclure dans notre composant les attributs et méthodes permettant à ces objets de se mettre à l'écoute :

```
public class SimpleDragAndDropPanel extends AbsolutePanel
{
    ...
    List listeners = new ArrayList();

    public void addDragDropListener(DragDropListener listener) {
        listeners.add(listener);
    }

    public void removeDragDropListener(DragDropListener listener) {
        listeners.remove(listener);
    }
}
```



```
}
}
```

Cette implémentation de la gestion des « listeners » est très classique et n'amène pas de remarques particulières. Il nous reste à faire le principal : capter les événements de dépôt (drop) d'une widget et diffuser cet événement. Ici nous avons un petit problème. L'événement est détecté dans le code propre au mécanisme de « glisser-déposer » de Yahoo. C'est-à-dire qu'il est émis par du code JavaScript extérieur à notre composant. Il faut donc intercepter cet événement en utilisant JavaScript. Cette interception devant ensuite appeler du code ... JAVA ! Heureusement, tout cela est possible avec GWT. C'est dans la méthode « addDraggableSlot », remaniée pour l'occasion que nous allons implémenter tout cela :

```
public class SimpleDragAndDropPanel extends AbsolutePanel
{
    ...
    public static native void addDraggableSlot(
        Element e,
        SimpleDragAndDropPanel ddp,
        String affordance,
        int counter)
    /*{
        e.id = 'drag_'+counter;
        var slot = new $wnd.YAHOO.example.DDPlayer(e.id, affordance);
        var fct = function(el, et) {
            ddp.@com.objetdirect.school.evaluator.client.SimpleDragAnd
```



```
DropPanel::dispatchDrop(Lcom/google/gwt/user/client/Element;Lcom/google/gwt/user/client/Element;)(el, et);
    return 1;
};
slot.subscribe(fct);
}*/;
...
}
```

Que se passe-t-il ici ? Nous enregistrons auprès de l'objet DDPlayer associé à la widget glissante, une fonction JavaScript réflexe. C'est la raison d'être de la dernière ligne de la méthode (slot.subscribe(fct);). Cette fonction réflexe (fct) est construite au sein de «addDraggableSlot». Elle reçoit en paramètre les avatars JavaScript des widgets concernées par l'opération de « glisser-déposer » : « el » est l'avatar de la widget glissante, et « et » est l'avatar de la widget de dépôt. La méthode fct se contente d'appeler une méthode JAVA de notre panel.

Observons l'appel :

```
dcp.@com.objetdirect.school.evaluator.client.SimpleDragAndDropPanel:
:dispatchDrop(Lcom/google/gwt/user/client/Element;Lcom/google/gwt/user/client/Element;)(el, et);
```

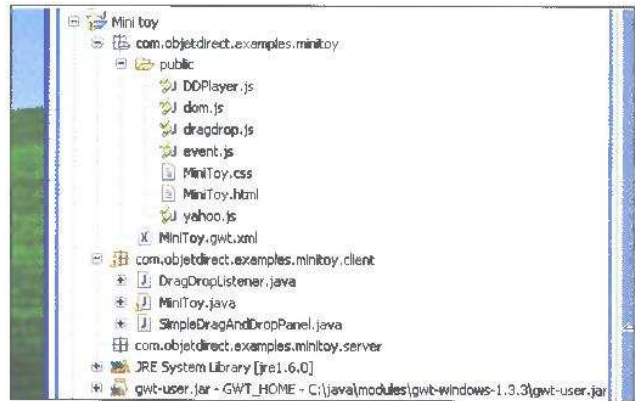
Attention : pour des raisons de lisibilité, la signature de la méthode invoquée apparaît sur plusieurs lignes. Dans le code, il est impératif que l'ensemble de la signature figure sur la même ligne.

dcp est le paramètre passé à la méthode « addDraggableSlot » qui référence notre composant panel. Pour identifier la méthode à appeler on doit donner sa signature in extenso : nom complet du package, nom de la classe et nom de la méthode. Cette définition se termine en donnant la liste des types de paramètres attendus. Noter que pour chaque type de paramètre, on renseigne son nom complet (avec le nom du package donc) précédé par « L » et en remplaçant les points intermédiaires par « / ». Il suffit de le savoir ...

La méthode appelée par notre méthode réflexe est « dispatchDrop » dont l'implémentation est donnée maintenant :

```
public class SimpleDragAndDropPanel extends AbsolutePanel
{
    ...
    public void dispatchDrop(Element e, Element t) {
        Widget draggable = getWidget(e);
        Widget target = getWidget(t);
        for (int i=0; i<listeners.size(); i++)
            ((DragDropListener)listeners.get(i)).onDrop(
                draggable, target);
    }
    ...
}
```

L'implémentation de cette méthode est évidente : la méthode « onDrop » de tous les objets à l'écoute est invoquée. Le point à remarquer est qu'en entrée, cette méthode ne dispose que des avatars JavaScript des widgets impliquées dans l'opération de « glisser-déposer ». Or ce sont les widgets elles-mêmes qui sont utiles en JAVA, pas les avatars JavaScript ! La classe des avatars : Element, ne permet aucune opération : ce



n'est qu'une coquille pour recouvrir des objets JavaScript complètement opaques pour JAVA. Son API est réduite à celle de la classe Object ! S'il est possible de retrouver l'avatar d'une widget (à l'aide de la méthode Widget.getElement()), le contraire n'est pas vrai. C'est à nous d'établir le lien partant d'un élément pour retrouver une widget. Pour cela, notre panel va contenir une Map, dont les clés sont les avatars et les valeurs sont les widgets propriétaires. La méthode getWidget exploite cette map :

```
public class SimpleDragAndDropPanel extends AbsolutePanel
{
    ...
    Map widgets = new HashMap();

    public Widget getWidget(Element e) {
        return (Widget)widgets.get(e);
    }
    ...
}
```

Enfin, les méthodes « add(Widget w, int left, int top) » et « remove(Widget w) » sont redéfinies afin que toute widget ajoutée ou retirée du panel vienne s'inscrire ou se dés-inscrire auprès de la Map :

```
public class SimpleDragAndDropPanel extends AbsolutePanel
{
    ...
    public void add(Widget w, int left, int top) {
        super.add(w, left, top);
        widgets.put(w.getElement(), w);
    }

    public boolean remove(Widget w) {
        widgets.remove(w.getElement());
        return super.remove(w);
    }
    ...
}
```

Notons que les méthodes add et remove présentées ici peuvent être utilisées directement sans dommage pour notre composant : les widgets ainsi confiés au panel ne seront pas prises en compte par le mécanisme

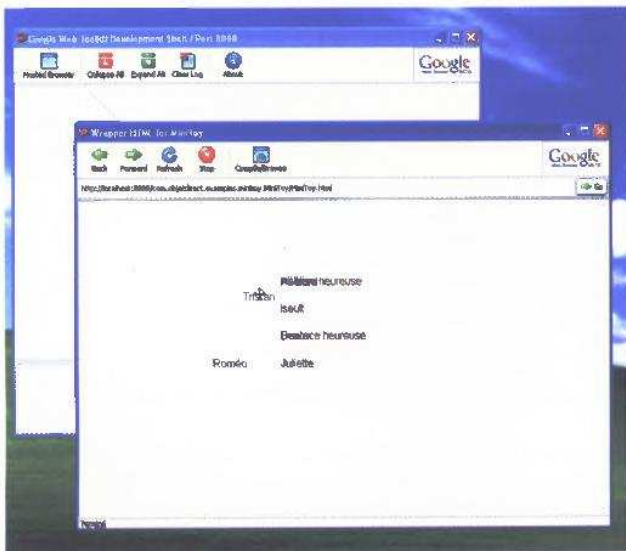
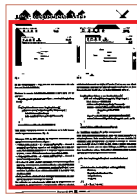


Fig. A

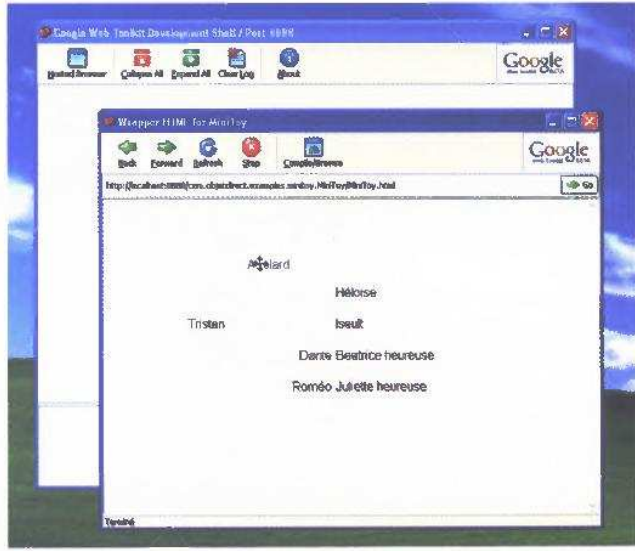


Fig. B

me de « Glisser-déposer ». Cela peut être une fonctionnalité très utile dans certaines circonstances.

Illustrons la nouvelle fonctionnalité apportée à notre panel par un exemple :

```
SimpleDragAndDropPanel amoursCelebres = new SimpleDragAndDropPanel();
...
DragDropListener listener = new DragDropListener() {
    public void onDrop(Widget draggable, Widget target) {
        Label dulcinee = (Label)target;
        dulcinee.setText(dulcinee.getText()+" heureuse");
    }
};
amoursCelebres.addDragDropListener(listener);
```

Vous pouvez maintenant constater la satisfaction de la belle lorsque vous lui apportez son amoureux. (Fig : A).

Notez que l'API du GWT propose les méthodes nécessaires pour connaître ou fixer la position des widgets :

- « Widget.getAbsoluteLeft » et « Widget.getAbsoluteTop » donnent la position d'une widget par rapport au coin haut/gauche de la page courante. Pour connaître la position de la widget par rapport au contenu du panel, il faudra déduire de ces valeurs, la position du panel.
- « Widget.getOffsetWidth » et « Widget.getOffsetHeight » donnent la taille d'une widget en pixels.
- Si le « fond de panier » de notre composant est un panel de type AbsolutePanel, nous disposons aussi des méthodes « AbsolutePanel.getWidgetLeft(Widget) » et « AbsolutePanel.getWidgetTop(Widget) » pour connaître la position d'une widget relativement à la position haut gauche du panel.
- Enfin, la méthode « AbsolutePanel.setWidgetPosition(Widget, int, int) » permet de déplacer une widget sur le panel.

Ces méthodes ne rentrent pas en conflit avec le fonctionnement du mécanisme de « glisser-déposer ». Vous pouvez les utiliser au sein de la

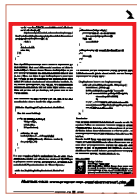
méthode « doDrop » de vos objets à l'écoute. C'est ce que nous allons faire dans notre exemple d'utilisation afin que le nom du galant se place à gauche de celui la belle pour que le tout reste lisible ! (Fig. B).

```
DragDropListener listener = new DragDropListener() {
    public void onDrop(Widget draggable, Widget target) {
        Label dulcinee = (Label)target;
        dulcinee.setText(dulcinee.getText()+" heureuse");
        int posX = amoursCelebres.getWidgetLeft(draggable)
            -draggable.getOffsetWidth()
            -5;
        int posY = amoursCelebres.getWidgetTop(draggable);
        amoursCelebres.setWidgetPosition(draggable, posX, posY);
    }
};
amoursCelebres.addDragDropListener(listener);
```

### La troisième version de notre composant

Nous allons maintenant permettre de « fixer » une widget pour qu'elle ne soit plus glissante. Cette nouvelle fonctionnalité va nous permettre de compléter notre compréhension du principe d'échange de données entre Javascript et Java à l'aide de JSNI. Jusqu'à présent nous avons ignoré dans le code Java les objets DDTarget et DDPlayer construits dans les méthodes natives. Désormais, il va falloir récupérer les objets DDPlayer afin de pouvoir plus tard les invalider :

```
public class SimpleDragAndDropPanel extends AbsolutePanel
{
    ...
    public static native JavaScriptObject addDraggableSlot(
        Element e,
        SimpleDragAndDropPanel ddp,
        String affordance,
        int counter)
    /*{
        e.id = 'drag_'+counter;
```



```

var slot = new $wnd.YAHOO.example.DDPlayer(e.id, affordance);
var fct = function(el, et) {
    ddp.@com.objetdirect.school.evaluator.client.SimpleDragAnd
DropPanel::dispatchDrop(Lcom/google/gwt/user/client/Element;Lcom/
google/gwt/user/client/Element;)(el, et);
    return 1;
};
slot.subscribe(fct);
return slot ;
}*/;
...
}

```

Notre objet DDPlayer est renvoyé sous la forme d'un objet Java de type JavaScriptObject. Nous avons déjà rencontré une classe qui dérive de JavaScriptObject : Element. Comme Element, JavaScriptObject ne propose aucune méthode autre que celles proposée par la classe de base de Java : Object. Les objets de type JavaScriptObject ne peuvent pas être manipulés en Java, ils ne peuvent qu'être récupérés depuis JavaScript et être transmis plus tard vers d'autres méthodes JavaScript. En fait, seuls les types simples (int, long, float, etc. mais aussi String) sont reconnus des deux côtés de JSNI. Les autres objets sont eux manipulables soit par Java, soit par JavaScript, mais jamais dans les deux mondes à la fois.

Nous conservons les objets DDPlayer dans une Map, ce qui nous permettra de les retrouver à partir de la widget associée :

```

public class SimpleDragAndDropPanel extends AbsolutePanel
{
    Map slots = new HashMap();

    public void addDraggableWidget(
        Widget widget,
        int left, int top,
        String affordance)
    {
        add(widget, left, top);
        JavaScriptObject dd = addDraggableSlot(
            widget.getElement(), affordance, counter++);
        slots.put(widget, dd);
    }
    ...
}

```

Il ne nous reste plus qu'à écrire la méthode qui fige une widget. Elle se décline en deux parties : une méthode Java qui retrouve l'objet DDPlayer et qui le passe à l'autre méthode, JavaScript celle-là. La méthode JavaScript retire le DDPlayer du mécanisme de « Glisser- déposer » :

```

public class SimpleDragAndDropPanel extends AbsolutePanel
{

```

```

...
public void unset(Widget widget) {
    JavaScriptObject dd = (JavaScriptObject)
        slots.remove(widget.getElement());
    if (dd!=null)
        removeSlot(dd);
}

public static native void removeSlot(JavaScriptObject dd)
/*-{
    dd.unreg();
}*/;
...
}

```

Il ne nous reste plus qu'à invoquer cette méthode dans notre exemple d'utilisation lorsque le galant a retrouvé sa belle, pour que l'on ne puisse plus séparer les amants réunis :

```

DragDropListener listener = new DragDropListener() {
    public void onDrop(Widget draggable, Widget target) {
        Label dulcinee = (Label)target;
        dulcinee.setText(dulcinee.getText()+" heureuse");
        int posX = amoursCelebres.getWidgetLeft(draggable)
            -draggable.getOffsetWidth()
            -5;
        int posY = amoursCelebres.getWidgetTop(draggable);
        amoursCelebres.setWidgetPosition(draggable, posX, posY);
        amoursCelebres.unset(draggable);
    }
};

```

### Conclusion

Nous disposons maintenant d'un composant à peu près complet. Nous pourrions l'affiner en interdisant l'utilisation directe de certaines méthodes héritées. Une autre option intéressante est de permettre de changer de type de panel et proposer un « fond de panier » dont la stratégie de placement soit plus élaborée que celle offerte par AbsolutePanel.

En moins de 100 lignes, nous avons écrit un composant complet permettant d'effectuer des opérations de « glisser-déposer » que nous pouvons piloter par programmation. Gageons que dans une version ultérieure du GWT, un tel composant sera proposé en standard. Cet exercice peut être reproduit facilement pour implémenter d'autres facilités AJAX comme les effets, tracés de graphiques, etc.

### ■ Henri DARMET



Directeur Technique

Objet Direct / Homsys Group

Objet Direct, filiale à 100% de Homsys Group est une société de conseil, de services et de formation, spécialisée sur les technologies objet et Web. Conseil en méthodologie, en architecture et en urbanisation du SI, développement applicatif, édition et distribution de logiciels. [www.objetdirect.com](http://www.objetdirect.com)