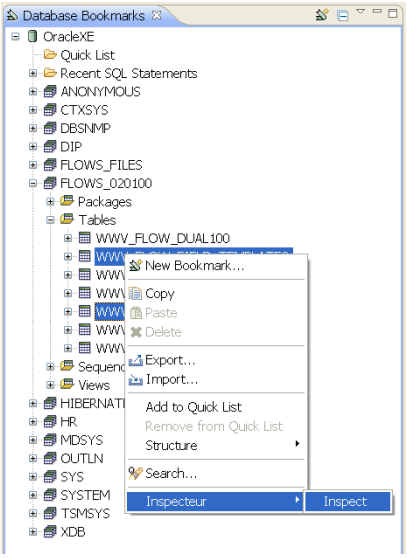
Inspektor für Inspektoren

Thomas Kestler, OnOffshoreSoft GmbH

Die Eclipse IDE ist sehr populär und Ihre interne Architektur erlaubt es neue Plugins zu ergänzen oder bestehende Plugins zu erweitern. Wenn man ein bestehendes Plugin erweitern will, kommt dem Entwickler jede Hilfe gerade recht.

Wir planen die Erstellung des JUDIE Plugin für Eclipse, welches Daten aus JDBC Datenbanken exportiert und importiert. Da wir das Rad nicht neu erfinden wollten, sollte JUDIE als eine Erweiterung des bestehenden Plugins wie z.B. QuantumDB entwickelt oder Data Tools Platform (DTP) werden. Eclipse ermöglicht es über sogenannte Extension Points neue Funktionalotät hinzuzufügen, hier verwenden wir neue Einträge im PopUp-Menu des `BookmarkView` von QuantumDB. Dies erfolgt über eine Object Contribution (ein weiterer Artikel über JUDIE selbst wird später folgen).

Das Problem bestand darin, zu verstehen, in welcher Form die QuantumDB `BookmarkView` die Informationen über die selektierten Knoten (Tabelle, Schemas, etc) liefert. Der Benutzer soll in der Lage sein, einzelne Tabelle (auch aus verschiedenen Schemata) oder Schemata zu selektieren und dann zu exportieren (oder importieren). Aus technischer Sicht besteht die `BookmarkView` aus einem `TreeViewer` und dieser hält Daten in Form von TreeNodes bzw. deren Unterklassen. Wenn der Benutzer nun eine Selektion vorimmt, kann diese Selektion in Form einer `IStructuredSelection` ausgelesen werden.



Um in JUDIE ein oder mehrere Tabellen exportieren zu können, müssen wir die Information haben, welche es selektiert wurden. Woher bekommen wir die Information über Tabellenname, Schema, Connections? Wussten wir auch nicht, darum haben wir Inspecteur geschrieben.

Erschwerend kommt hinzu, dass die BookmarkView ihre eigene Art hat, die Selektion zu liefern. Der Standard sieht vor, dass man über den `IWorkbenchPart` (getSite()) den `ISelectionProvider` bekommt, der dann die `Iselection` liefert. Hier liefert `getSelectionProvider()` allerdings null,da nie gesetzt.

Wir wollten verstehen, wie die Daten in der `BookmarkView` strukturiert sind. Also haben wir das Inspecteur Plugin gebaut, welches die aktuelle Selektion analysiert und dem Benutzer als Baum anzeigt. Wie oben erwähnt, kann man Menueinträge zu jeder View oder jedem Objekt hinzufügen. Inspecteur hängt seinen Menueintrag an alle Objekte der Klasse java.lang.Object (also alles):

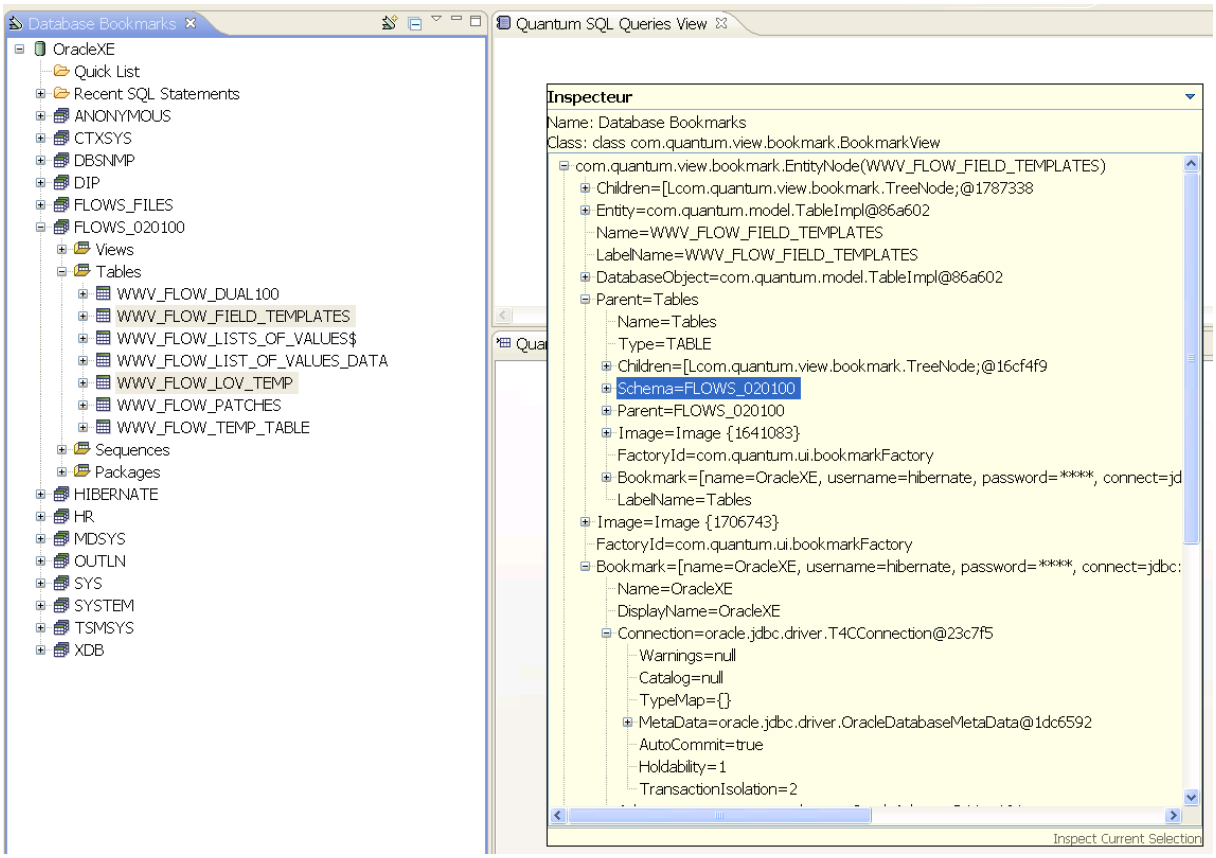
```
<objectContribution
      id="JUDIEclipse.contribution1"
      objectClass="java.lang.Object">
   <menu
         id="JUDIEclipse.menu1"
         label="Inspecteur"
         path="additions">
      <separator
            name="group1">
      </separator>
   </menu>
   <action
         class="judieclipse.popup.actions.InspectAction"
         enablesFor="+"
         id="JUDIEclipse.newAction"
         label="Inspect"
         tooltip="Inspect selected items"
         menubarPath="JUDIEclipse.menu1/group1">
   </action>
</objectContribution>
```

Das ist eigentlich gar keine gute Idee und wir empfehlen den Einsatz auch nur während der Entwicklung eines Plugins. Das Lustige an Inspecteur ist nun aber, dass sein Menueintrag fast überall in der Workbench verfügbar ist, also auch im Package Explorer, Nagivator, Problems List, etc.

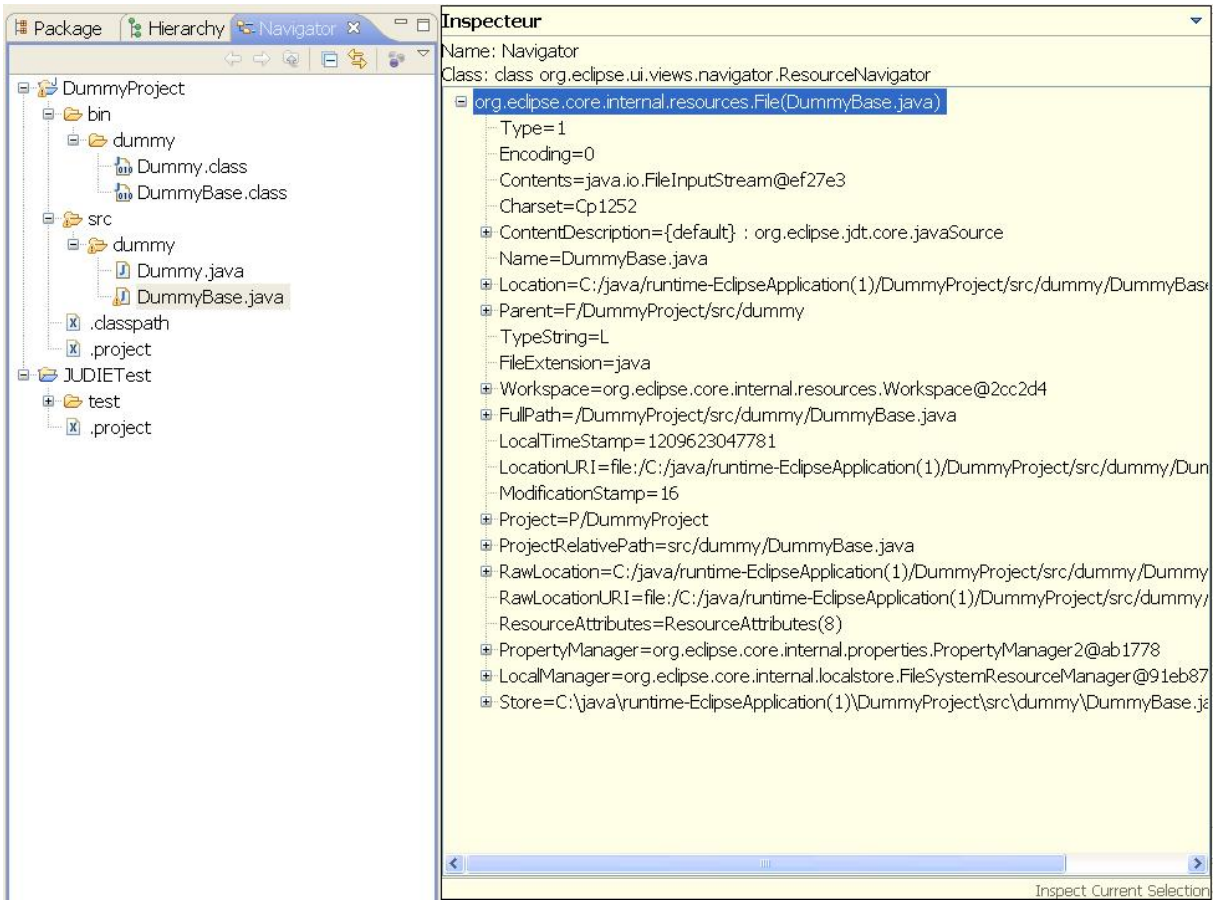
Der nächste Schritt, um Inspecteur zum Laufen zu bringen war es, ein `ActionDelegate` zu schreiben, welches die Selektion inspiziert. Da, wie oben gesagt, die Methode, wie die View die Selektion liefert, variieren kann, war das etwas trickreich (wir verwenden Reflection, um die View zu inspizieren und raten die passende Methode, welche die Selection oder Site liefert). Kurz gesagt, wir versuchen beides, den speziellen Fall und den Standardfall.

Nachdem wir die Selektion haben, müssen wir sie nur noch anzeigen. Dazu haben wir uns für einen PopUp-Dialog entschieden. Den Inhalt in einem eigenen View-Fenster anzuzeigen erschein uns nicht intuitiv. Der PopUp-Dialog verschwindet, wenn man woanders hin klickt.

Die Daten für den Baum (TreeViewer) werden immer erst auf Anforderung aufgebaut: klappt der Benutzer einen Knoten auf, so wird `ItreeContentProvider.getChildren()` aufgerufen und die Kinder mittels Reflection ermittelt. Das ist dringend nötig, denn im Baum können ja zyklische Referenzen existieren und dadurch wäre es unmöglich, die Bauminhalt im Voraus zu berechnen. Zudem ist dieses vorgehen ressourcensparender.



Das Plugin inspiziert nicht nur die QuantumDB BookmarkView, sondern auch andere Plugins und grundlegende Komponenten der Workbench:



Inspecteur kann eine Vielzahl verschiedener Selektionen inspizieren. Falls das bei einer bestimmten View nicht klappt, hält diese sich höchst wahrscheinlich nicht an das Standard Eclipse API. Vielleicht ist es aber auch ein Bug in Inspecteur, dann bitte eine Bug auf der Projektseite eintragen (bitte .metadata/.log mitsenden):

http://sourceforge.net/projects/inspecteur

Zielplatform ist Eclipse 3.3 und J2SE1.5. Ein Downgrade auf ältere Versionen oder JAVA 1.4 sollte prinzipiell möglich sein.

Ganz nebenbei: eigentlich sollte das Plugin ja „Inspector Clouseau“ lauten, aber der Name Clouseau ist markenrechtlich bereits geschützt und wir wollten Streit deswegen aus dem Weg gehen.