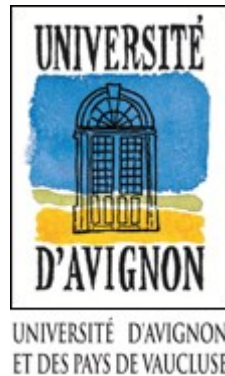


Prototypage et Interfaces utilisateurs



GL Avancé: Prototypage et interfaces utilisateur

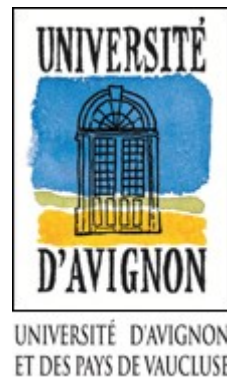
2 / 69

Objectifs du cours

- **Prototypage simple (dit horizontal):**
 - Conception d'une maquette statique (mockup)
 - Traduction des besoins utilisateur en composants graphiques
- **Prototypage fonctionnel (dit vertical):**
 - Ajout d'interactivité : maquette dynamique
 - Binding du modèle de données avec l'interface
 - Réalisation d'un scénario d'utilisation (use case)
- **Prototypage d'échelle sur une plateforme dédiée:**
 - Utilisation d'un environnement de fenêtrage de haut niveau
 - Découpage en plugins
 - Branding et personnalisation
 - Déploiement et livraison du prototype

3ème partie:

Prototypage avancé



GL Avancé: Prototypage et interfaces utilisateur

4 / 69

Prototypage sur une plateforme dédiée

- **Introduction:**

- Concept d'ingénierie: ne pas réinventer la roue, carrée
- Réutiliser un environnement éprouvé, optimisé et fiable
- Rester concentrer sur son domaine de compétence
- Erreur: prototypage avancé sans outillage



GL Avancé: Prototypage et interfaces utilisateur

5 / 69

Prototypage sur une plateforme dédiée

- **Introduction:**
 - Prototypage avancé d'expert bien outillé



GL Avancé: Prototypage et interfaces utilisateur

6 / 69

Prototypage sur une plateforme dédiée

- **Définition de la plateforme dédiée:**

- Environnement en charge du cycle de vie d'une application. Fournit les briques logicielles de base pour construire un client riche. Aussi appelée RCP (Rich Client Platform).

- **Intérêts d'un RCP:**

- Architecture robuste et fiable
- Ensemble de composants avancés pour une interface cohérente
 - ✓ Gestionnaire de fenêtre, Menus, Système de progression, Propriétés applicatives, ...
- Architecture modulaire pensée pour être adaptable aux besoins
- Services réutilisables et extensibles
- Mise à jour en ligne automatisée
- Internationalisation intégrée
- Distribution et déploiement simplifiée

GL Avancé: Prototypage et interfaces utilisateur

7 / 69

Prototypage sur une plateforme dédiée

- **Rich Client Platform Java:**

- Spring RCP:

- ✓ Java / Swing



- Eclipse RCP:

- ✓ Java / SWT / JFace / OSGi



- E(fx)clipse:

- ✓ Java / JavaFX / OSGi



- NetBeans Platform 8.1:

- ✓ Java / AWT / Swing / JavaFX / Module



GL Avancé: Prototypage et interfaces utilisateur

8 / 69

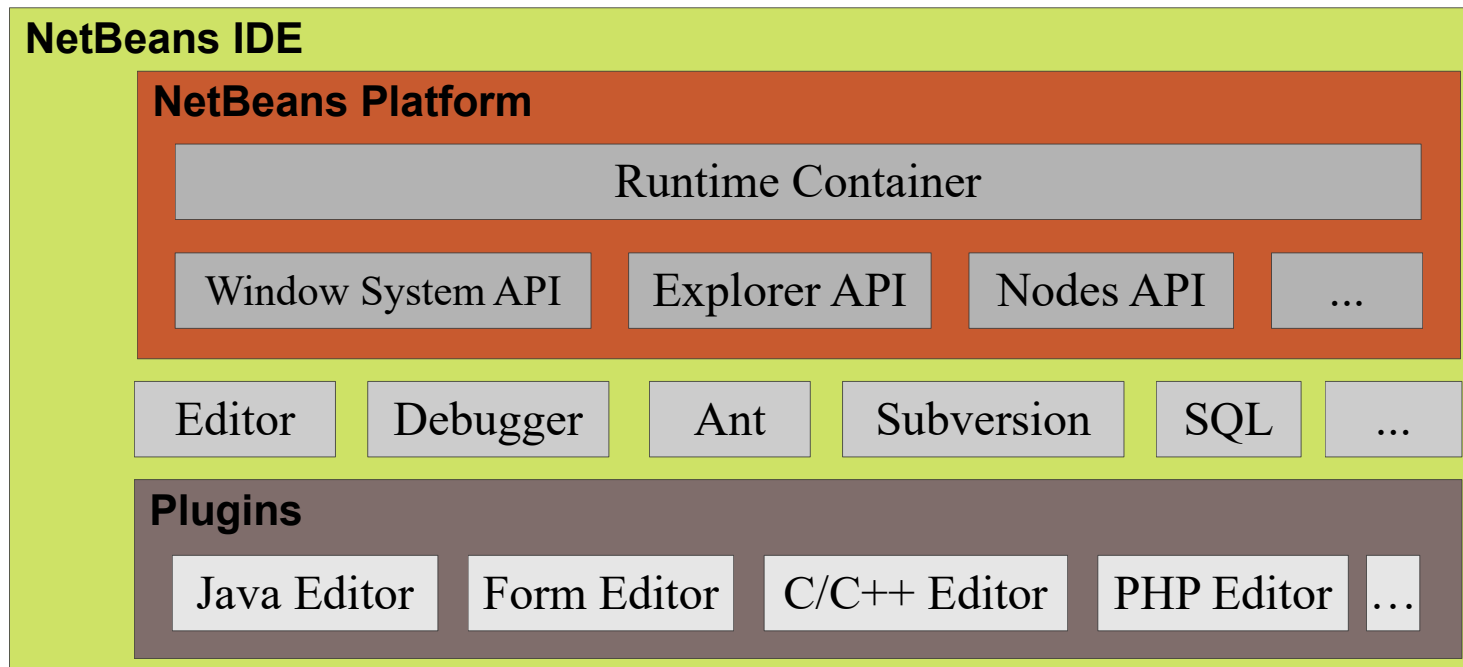
Architecture type d'une application

- **Socle de base : NetBeans Platform**

- ✓ Runtime Container: le noyau, pour une application de type serveur ou console
- ✓ Modules de la plateforme : couche graphique avec framework complet

- **Socle applicatif (exemple avec NetBeans IDE)**

- ✓ Modules spécifiques de l'application
- ✓ Plugins de l'application

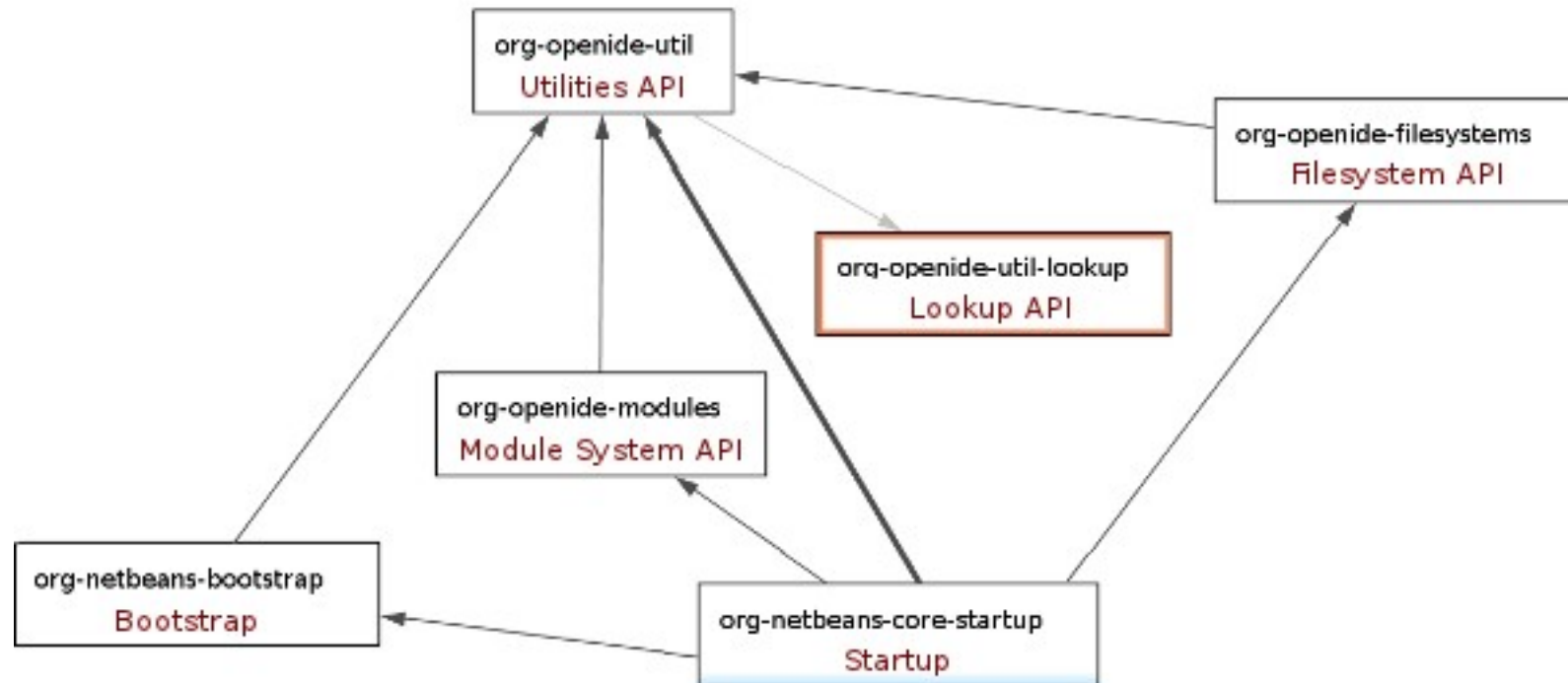


GL Avancé: Prototypage et interfaces utilisateur

9 / 69

Le runtime container : 6 modules

- ✓ Startup : main de l'application avec le code nécessaire au démarrage
- ✓ Bootstrap : chargement et composition des modules pour former l'application
- ✓ File System API : système de fichier virtuel avec accès orienté par flux
- ✓ Module System API : cycle de vie des modules de l'application (version, public API, OSGi, ...)
- ✓ Lookup API : mécanisme générique de communication inter modules avec couplage faible
- ✓ Utilities API : classes utilitaires partagées entre les modules du runtime container



GL Avancé: Prototypage et interfaces utilisateur

10 / 69

Les modules complémentaire de la plateforme

- **Framework:**

- ✓ Window System API : création de composants gérées par le système de fenêtrage de NetBeans
- ✓ Dialog API : création de notifications et de boîtes de dialogues avancées
- ✓ Action API : support des actions via les menus, les toolbars ou les raccourcis claviers
- ✓ Node API : modèle générique de représentation et de visualisation des données
- ✓ Progress API : visualisation de l'avancement de tâches longues
- ✓ Settings API : objets persistants pour l'enregistrement de propriétés accessibles via le lookup
- ✓ Text API : accès aux fonctionnalités d'édition, création de nouveaux éditeurs
- ✓ Java Help Integration, Option API, ...

- **Composants réutilisables:**

- ✓ Favorites : permet l'organisation de fichiers favoris
- ✓ Output Window : vue multi onglets affichant des messages de l'application (log, erreurs, ...)
- ✓ Auto Update Service : service de mise à jour automatique des modules
- ✓ ...

GL Avancé: Prototypage et interfaces utilisateur

11 / 69

Définition d'un module

- **Regroupement de classes apportant des fonctionnalités de différentes natures**
- **Exposition des paquetages accessibles, masquage de la structure interne**
- **Gestion explicite des dépendances entre les modules**
 - Nom des modules utilisés
 - Version des modules compatibles
- **Structuré dans une archive JAR contenant :**
 - Fichier Manifest (obligatoire)
 - Fichier Layer
 - Classes java
 - Ressources (icônes, internationalisation, propriétés, ...)

→ **Mécanisme d'extension dynamique de la plateforme par agrégation**

GL Avancé: Prototypage et interfaces utilisateur

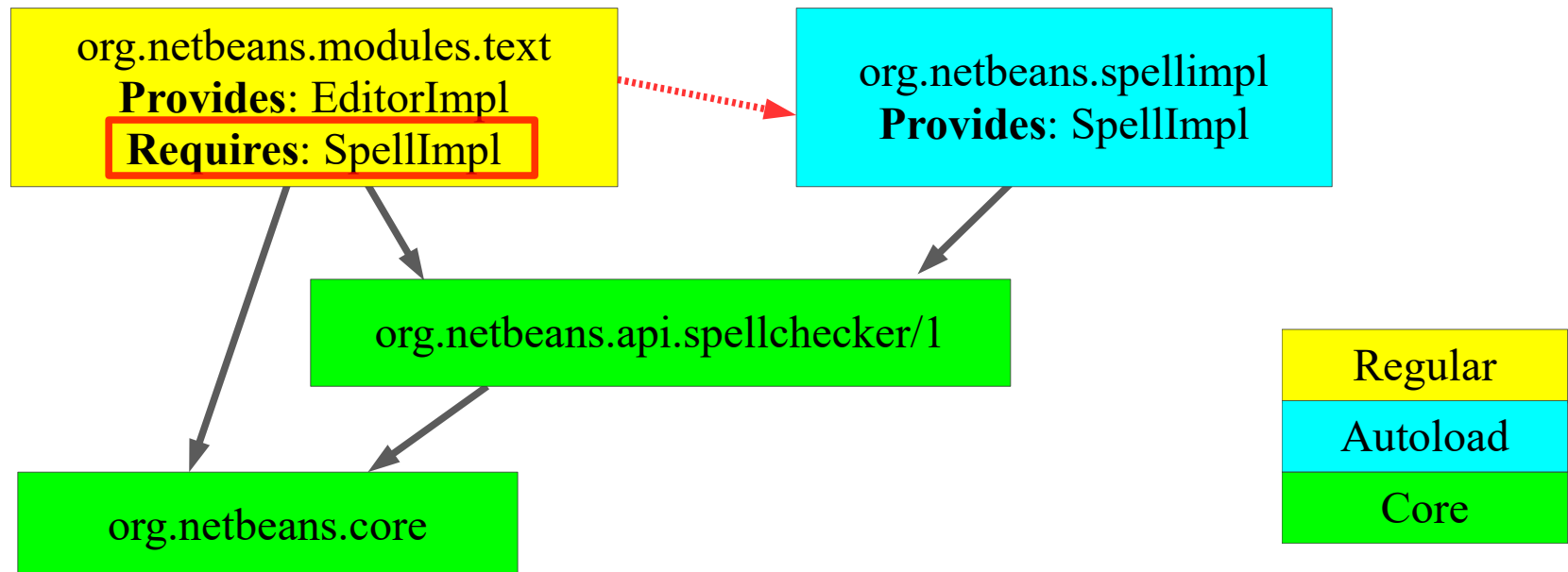
12 / 69

Principe de chargement du module

- **Types de module:**

- ✓ Déclaration dans le fichier « project.properties »
- ✓ Regular: chargement au démarrage de l'application (cas de la plupart des modules applicatifs)
- ✓ Autoload: activation à la demande (cas de la plupart des modules bibliothèques)
- ✓ Eager: uniquement si toutes les dépendances sont résolues pour accélérer le démarrage

- **Activation des modules au runtime:**



GL Avancé: Prototypage et interfaces utilisateur

13 / 69

Création d'un module

- **Description du module dans le fichier « manifest.mf » :**

- ✓ Identification unique du module :

```
Manifest-Version: 1.0  
OpenIDE-Module: org.netbeans.modules.text
```

- ✓ Spécification de la version :

```
OpenIDE-Module-Specification-Version: 1.13
```

- ✓ Déclaration des interfaces exposées :

```
OpenIDE-Module-Provides: EditorImpl
```

- ✓ Définition des dépendances du module :

```
OpenIDE-Module-Module-Dependencies:  
    org.netbeans.api.spellchecker/1 > 1.3,  
    org.netbeans.core > 4.32
```

- ✓ Déclaration des services requis :

```
OpenIDE-Module-Requires: SpellImpl
```

- ✓ Fichier de contribution à la plateforme NetBeans :

```
OpenIDE-Module-Layer: fr/ismart/netbeans/module/layer.xml
```


GL Avancé: Prototypage et interfaces utilisateur

14 / 69

Création d'un module

- ✓ Déclaration des modules amis :

```
OpenIDE-Module-Friends:  
org.netbeans.modules.text.friend
```

- ✓ Spécification de la version du compilateur Java:

```
OpenIDE-Module-Java-Dependencies: Java > 1.7
```

- ✓ Déclaration des modules optionnels:

```
OpenIDE-Module-Recommends: java.sql.Driver
```

- ✓ Définition des classes visibles depuis les autres modules :

```
OpenIDE-Module-Module-Dependencies:  
org.netbeans.modules.text.*,  
org.netbeans.modules.text.common.*
```

- ✓ Enregistrement d'une classe d'installation :

```
OpenIDE-Module-Install: Install.class
```

- ✓ Fichier d'internationalisation :

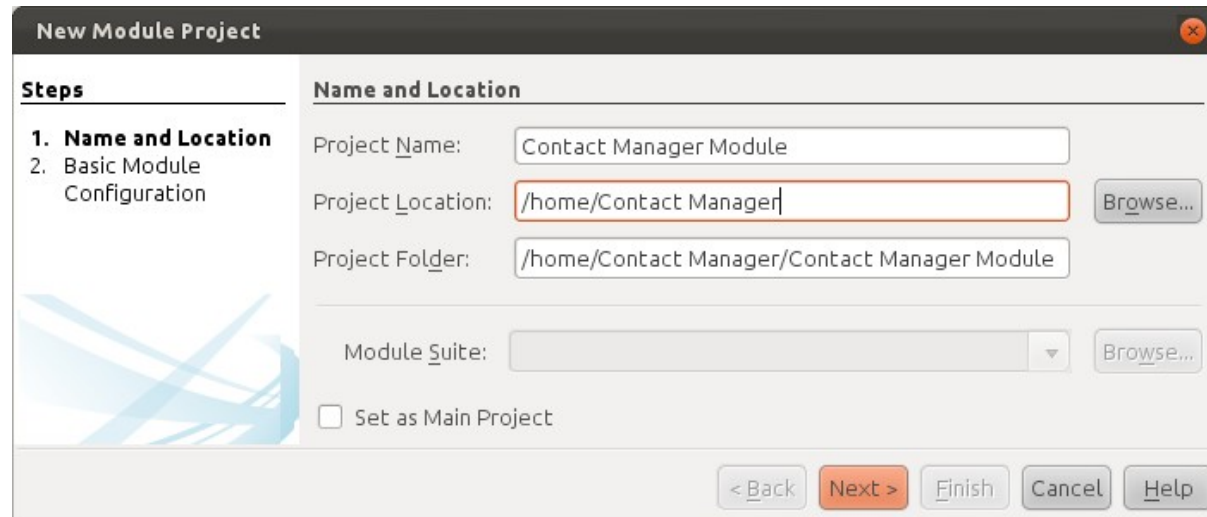
```
OpenIDE-Module-Localizing-Bundle:  
fr/ismart/netbeans/module/Bundle.properties
```

GL Avancé: Prototypage et interfaces utilisateur

15 / 69

Création d'un module depuis NetBeans

- Nouveau projet de type « module » :

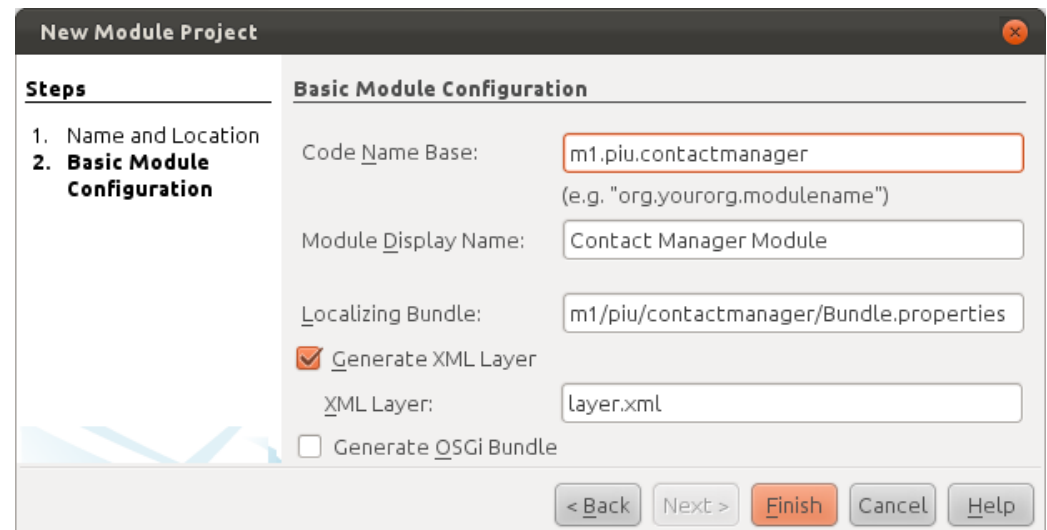


The screenshot shows the 'New Module Project' dialog box in NetBeans. The 'Steps' panel on the left indicates '1. Name and Location' is the current step. The 'Name and Location' panel contains the following fields: 'Project Name' (Contact Manager Module), 'Project Location' (/home/Contact Manager), 'Project Folder' (/home/Contact Manager/Contact Manager Module), and 'Module Suite' (empty). There are 'Browse...' buttons for the Location and Suite fields. A 'Set as Main Project' checkbox is at the bottom. Navigation buttons at the bottom include '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

- Description du module :

- ✓ Identifiant unique
- ✓ Nom du module
- ✓ Bundle d'internationalisation
- ✓ Création du layer.xml

- Génération automatique du manifest



The screenshot shows the 'New Module Project' dialog box in NetBeans, Step 2: Basic Module Configuration. The 'Steps' panel on the left indicates '2. Basic Module Configuration' is the current step. The 'Basic Module Configuration' panel contains the following fields: 'Code Name Base' (m1.piu.contactmanager), 'Module Display Name' (Contact Manager Module), 'Localizing Bundle' (m1/piu/contactmanager/Bundle.properties), 'Generate XML Layer' (checked), 'XML Layer' (layer.xml), and 'Generate OSGi Bundle' (unchecked). Navigation buttons at the bottom include '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

GL Avancé: Prototypage et interfaces utilisateur

16 / 69

Création d'un module depuis NetBeans

➤ Propriétés du projet « module » :

✓ Versioning

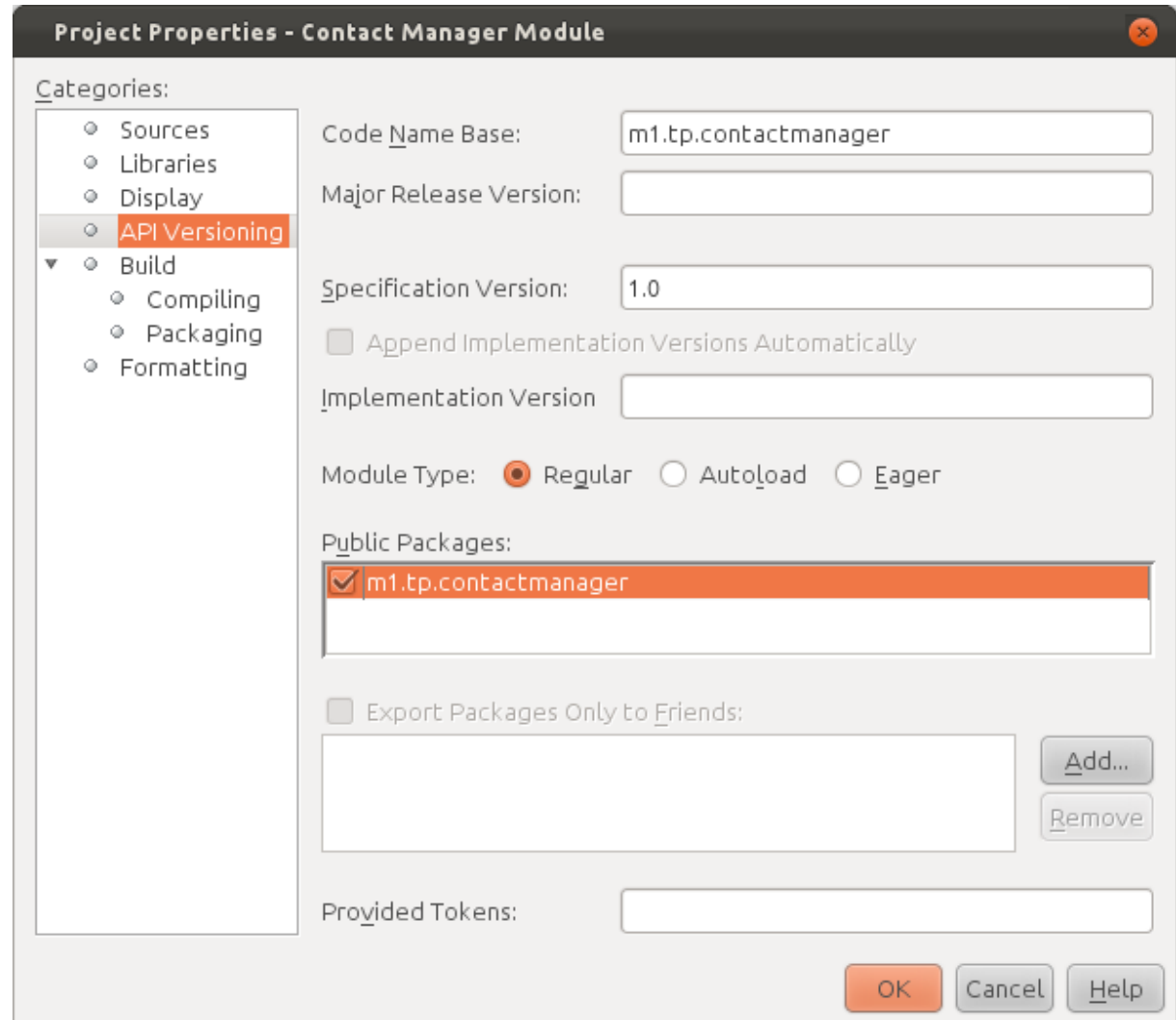
(fichier « manifest.mf »)

✓ Type de module

(fichier « project.properties »)

✓ Exposition des paquetages

(fichier « project.xml »)

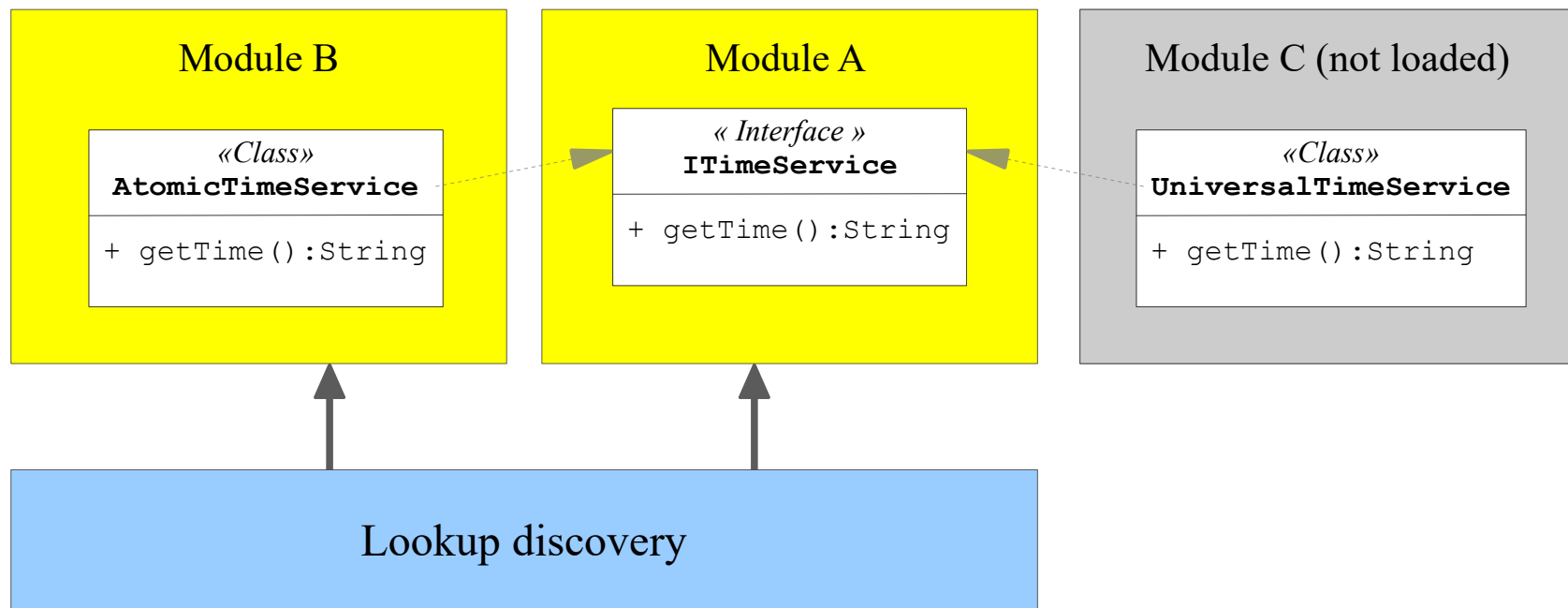


GL Avancé: Prototypage et interfaces utilisateur

17 / 69

Mécanisme de Lookup inter-modules

- **Couplage faible interface / implémentation(s) :**
 - Déclaration d'une interface exposant les signatures des méthodes requises
 - Autoriser une ou plusieurs implémentations
 - Conception dans des modules séparés
 - Aspect dynamique selon la présence du module dans les dépendances de l'application

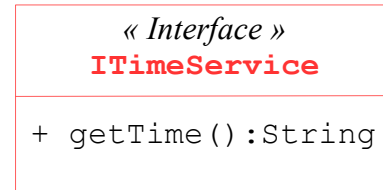


GL Avancé: Prototypage et interfaces utilisateur

Mécanisme de Lookup inter-modules

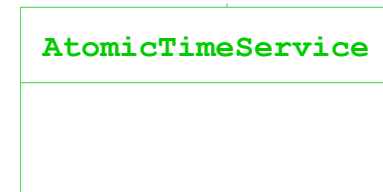
• Définition d'un service dans un paquetage public d'un **module A**

```
package fr.ismart.A.service ;
public interface ITimeService{
    public String getTime() ; }
```



• Enregistrement d'une implémentation du service dans un **module B**

```
package fr.ismart.B.service ;
public class AtomicTimeService
    implements fr.ismart.A.service.ITimeService{
    public String getTime() { [...] } }
```



- Créer le fichier META-INF/services/**fr.ismart.A.service.ITimeService**
 - ✓ Ajouter la ligne : **fr.ismart.B.service.AtomicTimeService**
- OU de façon automatique en annotant la classe implémentant le service **AtomicTimeService**:
 - ✓ `@ServiceProvider (service=ITimeService.class)`

• Découverte des instances du service (depuis un module dépendant de A)

```
Collection< ? extends ITimeService> services =
    Lookup.getDefault().lookupAll(ITimeService.class) ;
```


GL Avancé: Prototypage et interfaces utilisateur

19 / 69

Contribution à la plateforme NetBeans

- **Le layer:**
 - Point central de la configuration : ce que le module apporte à la plateforme
 - Système de fichier hiérarchique contenant des points d'extensions
 - Fichier au format XML : `layer.xml`
- **Points d'extensions principaux :**
 - **Actions** : définitions des instances de `javax.swing.Action`
 - **Menu** : personnalisations de la barre de menu
 - **Toolbars** : personnalisations de la barre d'outils
 - **Navigator/Panels** : ajout de panels dans la vue « navigator »
 - **OptionsDialog** : création d'un panel dans les options
 - **Services** : enregistrement de services (sans utiliser le lookup)
 - **Shortcuts** ou **Keymaps/NetBeans** : liste des raccourcis claviers
 - **TaskList** : liste des taches utilisateurs (TODO, Warnings, Errors, ..)
 - **Windows2** : configuration du système de fenêtrage

GL Avancé: Prototypage et interfaces utilisateur

20 / 69

Structure du fichier layer.xml

- **Balise « folder » :**

- Définition d'une contribution à un point d'extension
- Sous dossiers possibles pour structurer la contribution
- Balise « file » :
 - ✓ **.instance** : décrit un objet qui peut être instancié
 - ✓ **.shadow** : lien référence vers une instance, souvent utilisé avec les singletons
 - ✓ Balise « attr » :
 - Valeur d'une propriété (type : intvalue, boolvalue, stringvalue, urlvalue, methodvalue, newvalue, bundlevalue)
 - Possibilité de changer l'ordre du dossier lors de la lecture du layer

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE filesystem PUBLIC "-//NetBeans//DTD Filesystem 1.1//EN"
"http://www.netbeans.org/dtds/filesystem-1_1.dtd">
<filesystem>
  <folder name="Menu">
    <folder name="Edit">
      <file name="m1-piu-module-MyAction..instance">
        <attr name="position" intvalue="1"/>
      </file>
    </folder>
  </folder>
</filesystem>
```

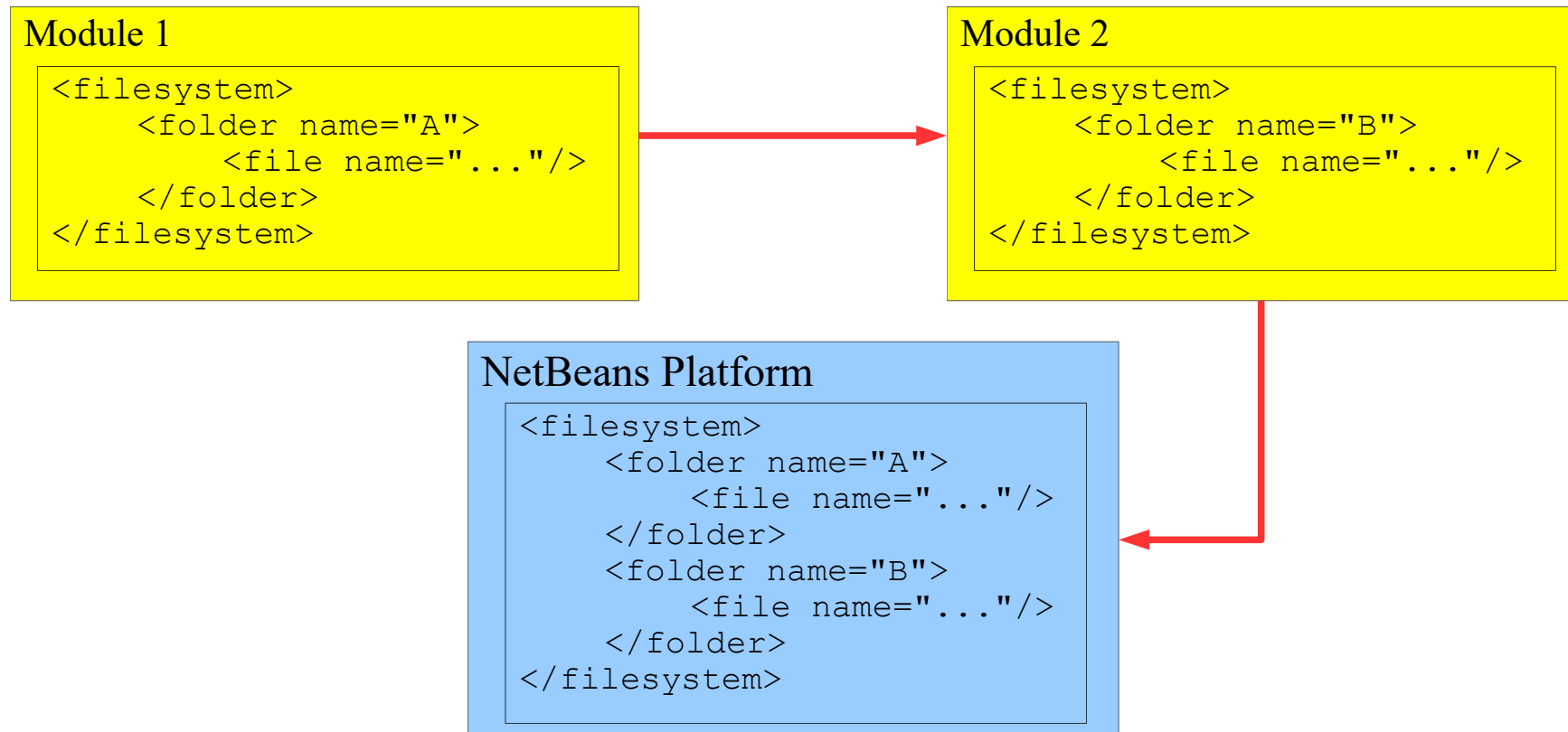
GL Avancé: Prototypage et interfaces utilisateur

21 / 69

Contribution à la plateforme NetBeans

- **Architecture des contributions en couches**

- Fichier layer.xml définit dans chaque module
- Chaque module spécifie ses propres contributions
- Agrégation des layers dans le System FileSystem de la plateforme au runtime



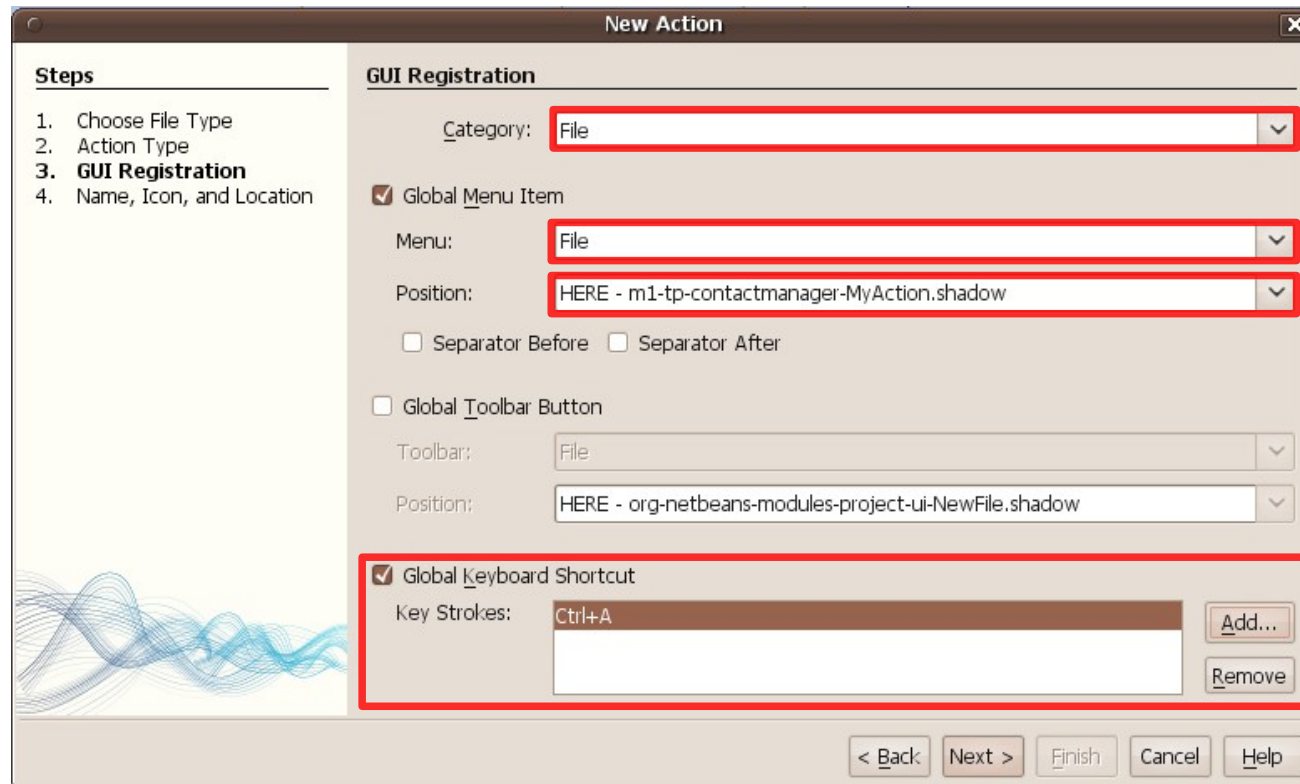
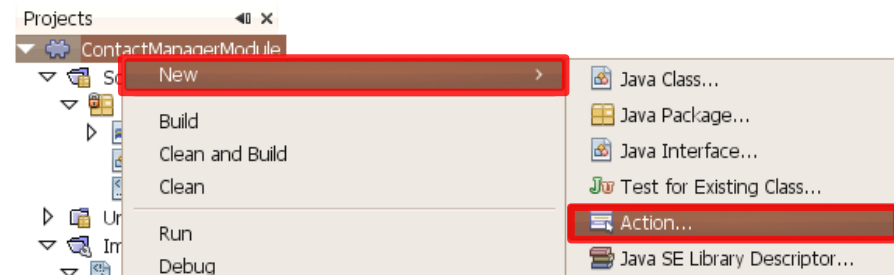
GL Avancé: Prototypage et interfaces utilisateur

22 / 69

Ajouter une action au menu

- **Utiliser l'assistant « New Action »:**

- ✓ Type d'action: `ActionListener`
- ✓ Intégration au menu
- ✓ Raccourci clavier



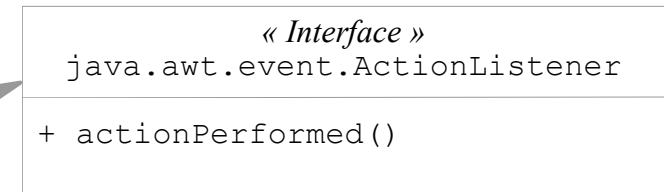
GL Avancé: Prototypage et interfaces utilisateur

23 / 69

Ajouter une action au menu

- **Code généré et annoté:**

- Classe « MyAction.class »:



```

@ActionID(
    category = "File",
    id = "m1.tp.contactmanager.MyAction"
)
@ActionRegistration(
    displayName = "#CTL_MyAction"
)
@ActionReference(path = "Menu/File", position = 1300)
@Messages("CTL_MyAction=My action")
public final class MyAction implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {
        // TODO implement action body
    }
}
  
```

- Clé d'internationalisation : `build/classes/[package]/Bundle.properties`:

- ✓ **CTL_MyAction** = My Action

GL Avancé: Prototypage et interfaces utilisateur

24 / 69

Ajouter une action au menu

- **Fichier généré par les annotations:**

- Mise à jour de la liste des actions : `build/classes/META-INF/generated-layer.xml`:

```
<folder name="Actions">
  <folder name="File">
    <file name="m1-tp-contactmanager-MyAction.instance">
      <attr name="delegate" newvalue="m1.tp.contactmanager.MyAction"/>
      <attr name="displayName" bundlevalue="m1.tp.contactmanager.Bundle#CTL_MyAction"/>
      <attr name="instanceCreate" methodvalue="org.openide.awt.Actions.alwaysEnabled"/>
      <attr name="noIconInMenu" boolvalue="false"/>
    </file>
  </folder>
</folder>
```

- Mise à jour de la liste des menu : `build/classes/META-INF/generated-layer.xml`:

```
<folder name="Menu">
  <folder name="File">
    <file name="m1-tp-contactmanager-MyAction.shadow">
      <attr name="originalFile" stringvalue="Actions/File/m1-tp-contactmanager-MyAction.instance"/>
    </file>
  </folder>
</folder>
```

GL Avancé: Prototypage et interfaces utilisateur

25 / 69

Ajouter une action au menu

- **Mise à jour automatique du contexte:**

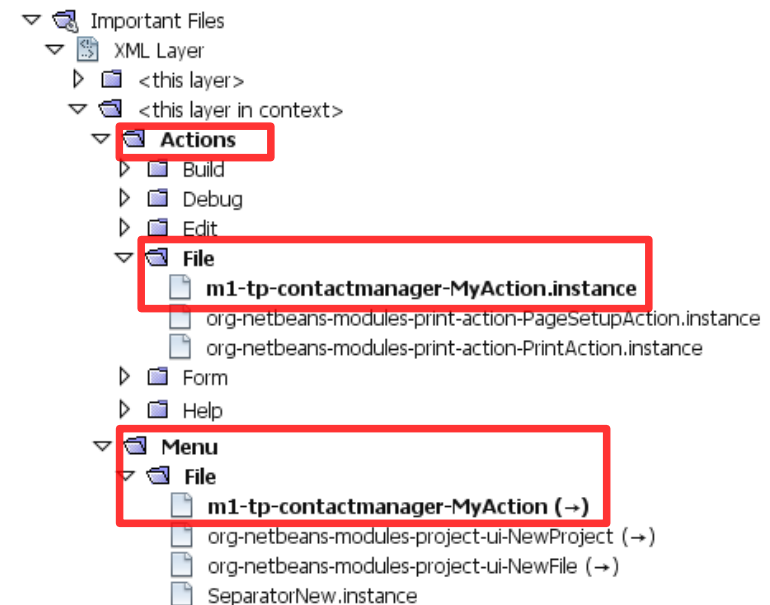
- ✓ Fichier « .instance »
- ✓ Lien « .shadow » vers l'instance

- **Modification manuelle des menus:**

- Supprimer les liens/folders du Layer

- **Position des menus pré-définis :**

- ✓ File: 100
- ✓ Edit: 200
- ✓ Navigate: 400
- ✓ Tool: 1000
- ✓ Help: 1300



GL Avancé: Prototypage et interfaces utilisateur

26 / 69

Ajouter une action à la barre d'outils

- Annotations spécifiques :**

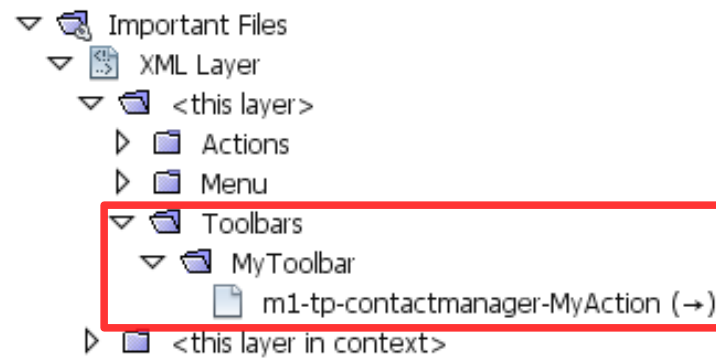
```
@ActionRegistration(
    iconBase = "m1/piu/my_action.png",
    displayName = "#CTL_MyAction"
)
@ActionReferences({
    @ActionReference(path = "Menu/File", position = 1200),
    @ActionReference(path = "Toolbars/File", position = 200)
})
```

- Fichier modifié par les annotations:**

```
<folder name="Toolbars">
  <folder name="MyToolbar">
    <file name="m1-tp-contactmanager-MyAction.shadow">
      <attr name="originalFile" stringvalue="Actions/File/m1-tp-contactmanager-MyAction.instance"/>
    </file>
  </folder>
</folder>
```

- Mise à jour du contexte:**

- Lien « .shadow » vers l'instance



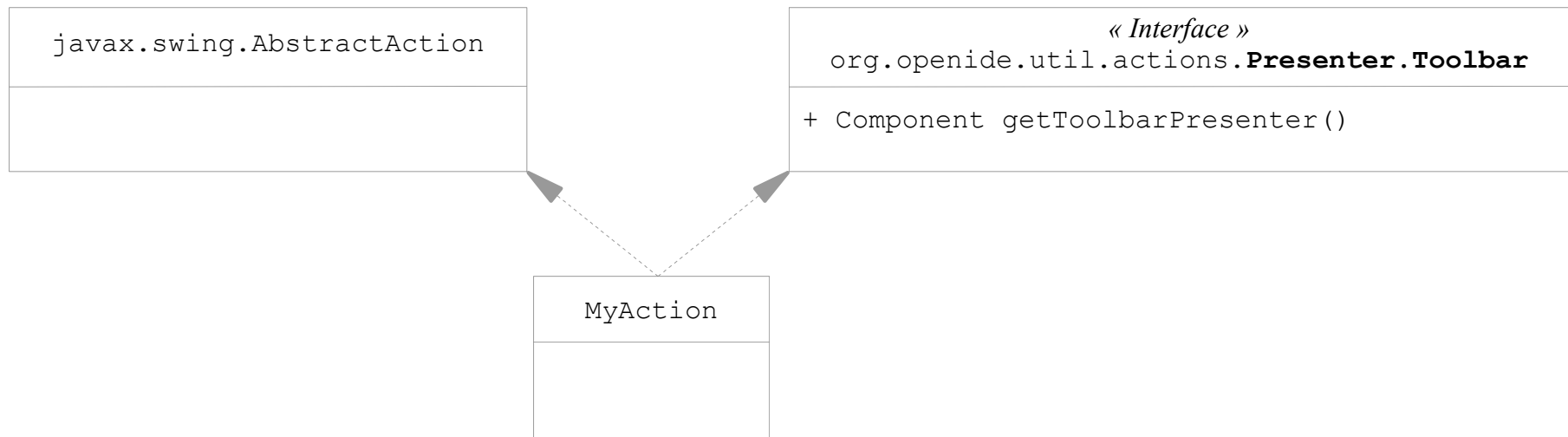
GL Avancé: Prototypage et interfaces utilisateur

27 / 69

Ajouter un composant à la barre d'outils

- **Modifier la classe action :**

- ✓ Implémenter l'interface `Presenter.Toolbar`
- ✓ Étendre la classe `AbstractAction`



```
public final class MyAction extends AbstractAction implements Presenter.Toolbar {

    @Override
    public Component getToolbarPresenter() {
        return new JLabel("Label in toolbar demo");
    }
}
```



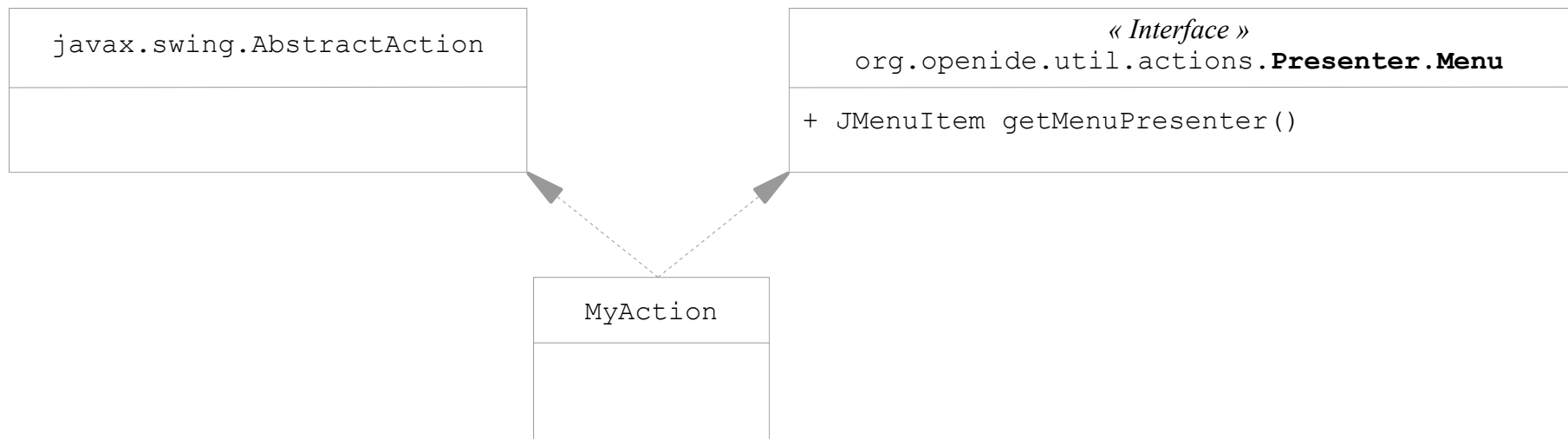
GL Avancé: Prototypage et interfaces utilisateur

28 / 69

Ajouter un item à la barre de menu

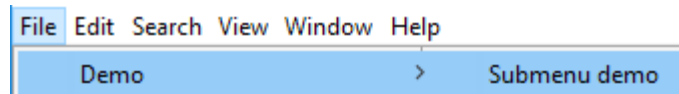
- **Modifier la classe action :**

- ✓ Implémenter l'interface `Presenter.Menu`
- ✓ Étendre la classe `AbstractAction`



```
public final class MyAction extends AbstractAction implements Presenter.Menu {

    public JMenuItem getMenuPresenter() {
        JMenuItem menuItem = new JMenuItem("Submenu demo");
        JMenu menu = new JMenu("Demo");
        menu.add(menuItem);
        return menu;
    }
}
```



GL Avancé: Prototypage et interfaces utilisateur

29 / 69

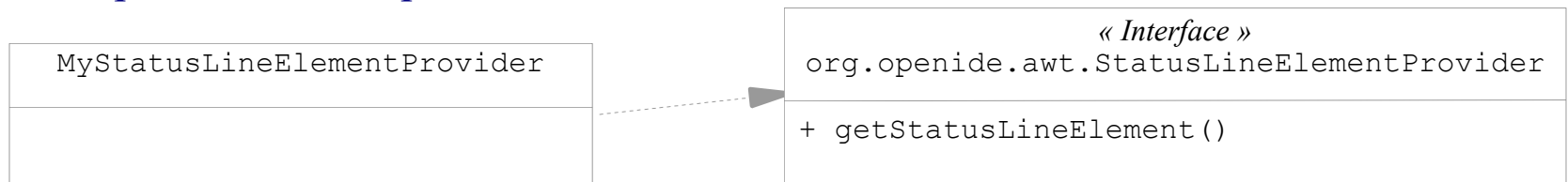
Barre de status

- **Écriture dans la barre standard:**

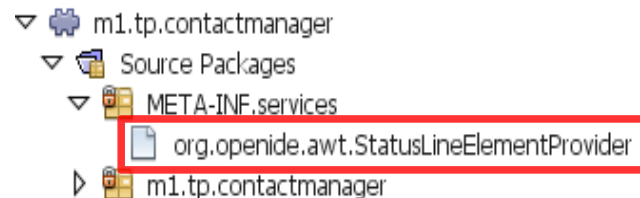
- Par lookup: `Lookup.getDefault().lookup(StatusDisplayer.class).setStatusText("Mon status");`
- Par accès statique: `StatusDisplayer.getDefault().setStatusText("Mon status");`

- **Barre d'état personnalisée:**

- Redéfinir le provider fourni par UI Utilities API:



- Créer le fichier `/src/META-INF/services/org.openide.awt.StatusLineElementProvider`:
 - ✓ Déclarer le provider dans le fichier: “`m1.tp.contactmanager.MyStatusLineElementProvider`”



GL Avancé: Prototypage et interfaces utilisateur

Le gestionnaire de fenêtres

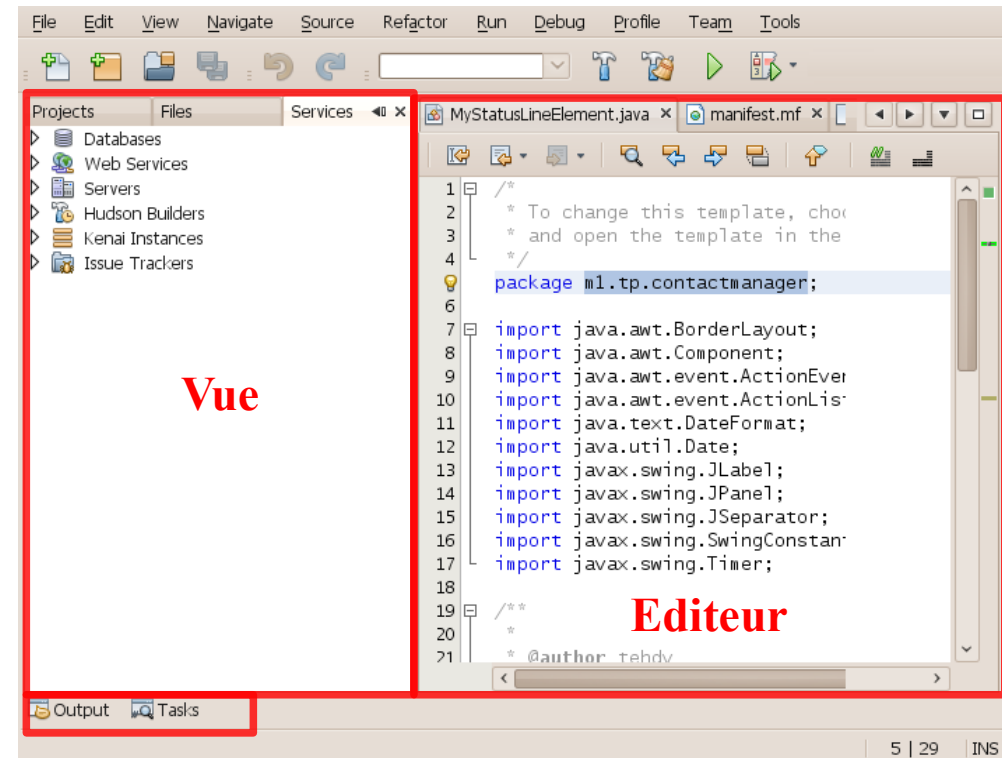
- **Les conteneurs:**

- **L'éditeur:**

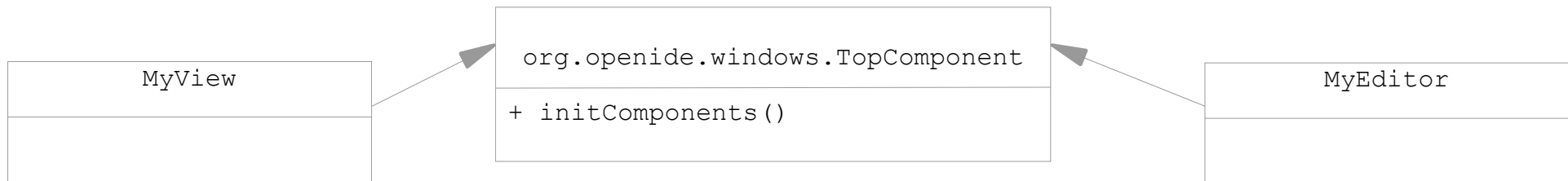
- ✓ Basé sur la notion de document
- ✓ Plusieurs documents éditables dans des onglets
- ✓ Affiché dans la zone centrale

- **La vue:**

- ✓ Composant situé autour de l'éditeur
- ✓ Existence unique: une instance par type de vue



- **Définition du conteneur au sens Java:**



GL Avancé: Prototypage et interfaces utilisateur

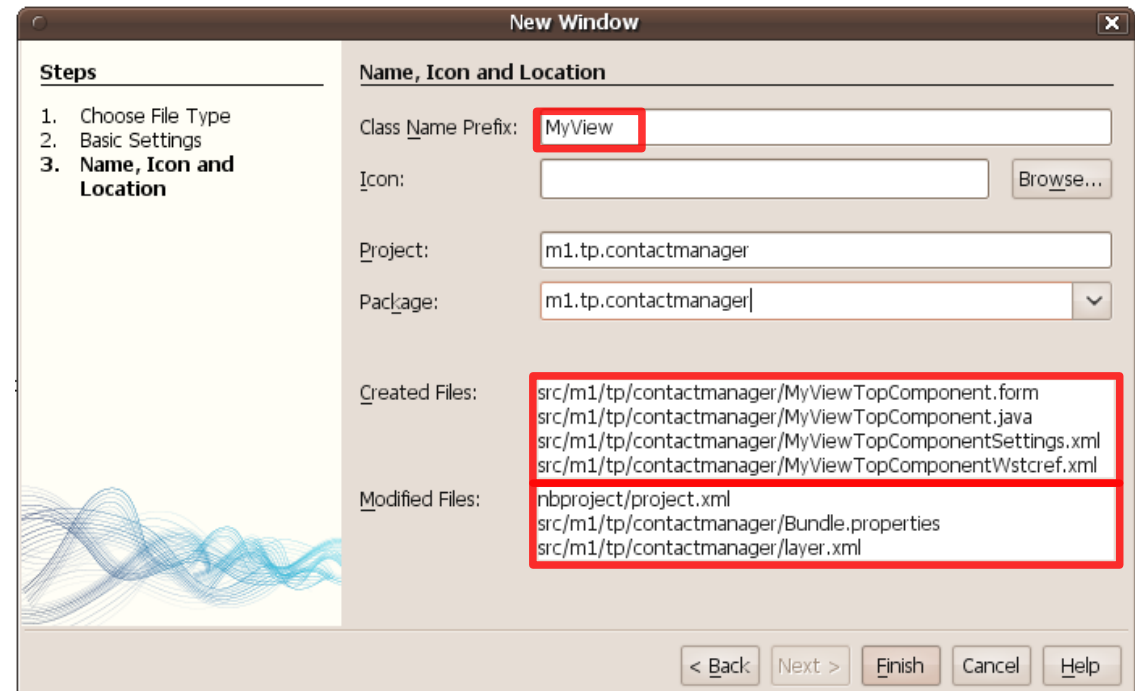
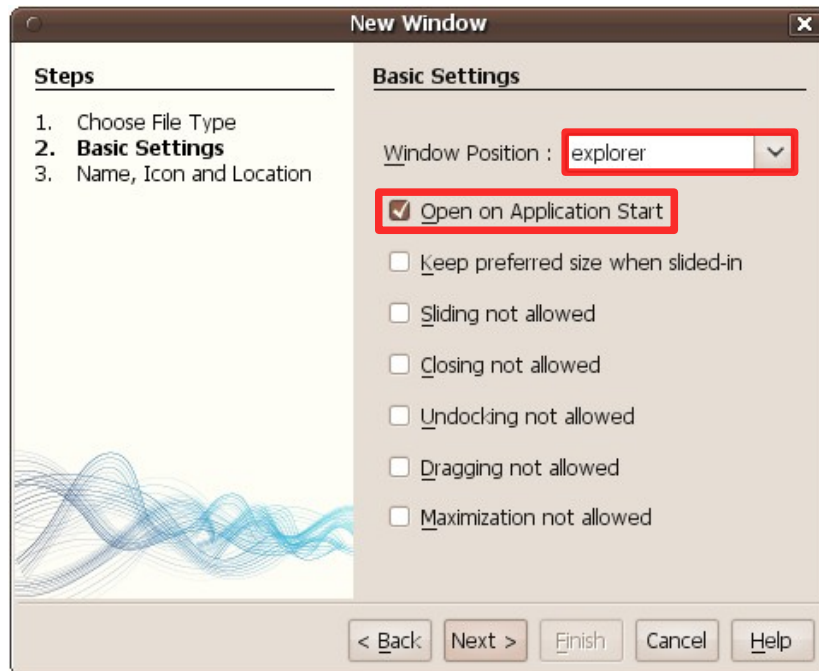
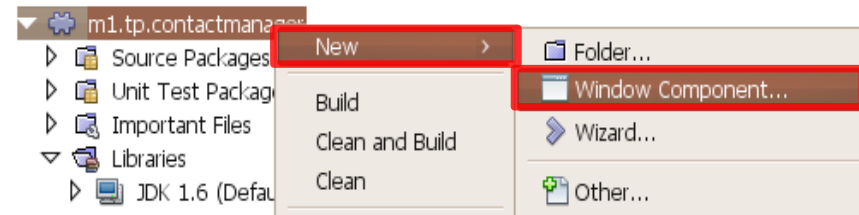
31 / 69

Le gestionnaire de fenêtres

- **Création d'un conteneur avec l'assistant:**

- **Choix du mode:**

- ✓ Editeur: editor (centre)
- ✓ Vue: explorer (gauche), output (bas), properties (droite), navigator (bas gauche)



GL Avancé: Prototypage et interfaces utilisateur

32 / 69

Le gestionnaire de fenêtres

- **Code généré:**

- Classe « MyViewTopComponent » pré-codée avec annotations

```
@ConvertAsProperties(
    dtd = "-//m1.tp.contactmanager//MyView//EN",
    autostore = false
)
@TopComponent.Description(
    preferredID = "MyViewTopComponent",
    //iconBase="SET/PATH/TO/ICON/HERE",
    persistenceType = TopComponent.PERSISTENCE_ALWAYS
)
@TopComponent.Registration(mode = "explorer", openAtStartup = true)
@ActionID(category = "Window", id = "m1.tp.contactmanager.MyViewTopComponent")
@ActionReference(path = "Menu/Window" /*, position = 333 */)
@TopComponent.OpenActionRegistration(
    displayName = "#CTL_MyViewAction",
    preferredID = "MyViewTopComponent"
)
@Messages({
    "CTL_MyViewAction=MyView",
    "CTL_MyViewTopComponent=MyView Window",
    "HINT_MyViewTopComponent=This is a MyView window"
})
public final class MyViewTopComponent extends TopComponent { [...] }
```

GL Avancé: Prototypage et interfaces utilisateur

33 / 69

Le gestionnaire de fenêtres

- **Création d'un mode personnalisé:**

- Fichier « MyModeWsmode.xml » :

```
<!DOCTYPE mode PUBLIC
"-//NetBeans//DTD Mode Properties 2.3//EN"
"http://www.netbeans.org/dtds/mode-properties2_3.dtd">
<mode version="2.3">
  <module name="ml.tp.contactmanager" spec="1.0"/>
  <name unique="MyMode"/>
  <kind type="view"/>
  <state type="joined"/>
  <constraints>
    <path orientation="vertical" number="0" weight="0.2"/>
    <path orientation="horizontal" number="0" weight="0.4"/>
  </constraints>
  <empty-behavior permanent="true"/>
</mode>
```

- Déclaration dans le layer:

```
<folder name="Windows2">
  <folder name="Modes">
    <file name="MyMode.wsmode" url="MyModeWsmode.xml"/>
  </folder>
</folder>
```

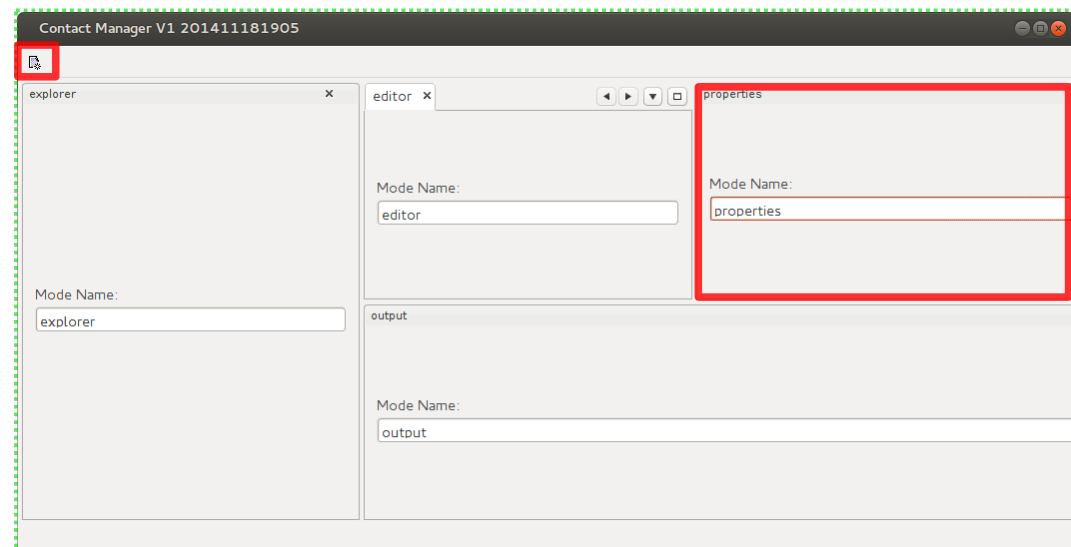
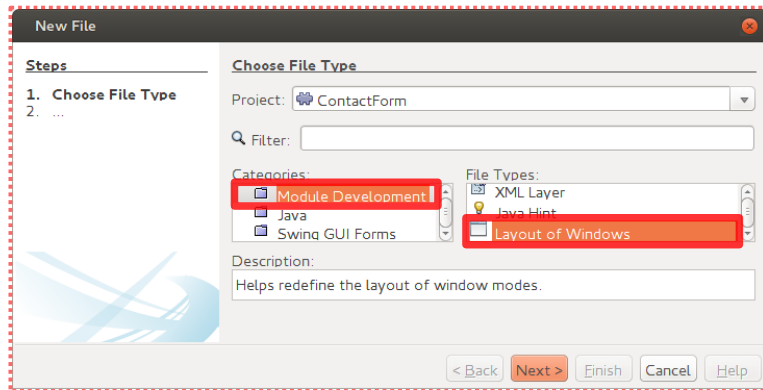
GL Avancé: Prototypage et interfaces utilisateur

34 / 69

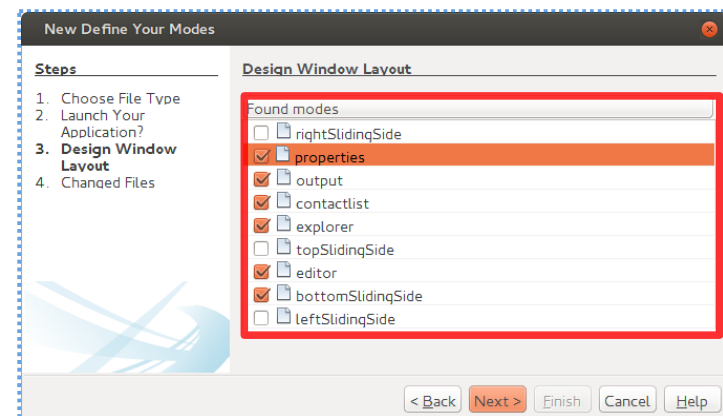
Le gestionnaire de fenêtres

- **Création d'un mode personnalisé avec l'assistant:**

- Depuis le menu « New File », dans la catégorie « Module Development », choisir le type « Layout of Windows » (*)



- Modifier les modes existants (*)
- Créer un nouveau mode (*)
- Sélectionner les mode à importer (*)
(après la fermeture de la fenêtre des modes)



GL Avancé: Prototypage et interfaces utilisateur

Les boites de dialogue (Dialog API)

- Message:**



```
NotifyDescriptor nd = new NotifyDescriptor.Message("Message", NotifyDescriptor.INFORMATION_MESSAGE);
```

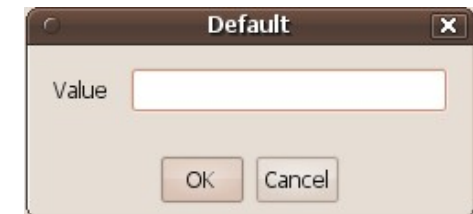
- Confirmation:**



```
NotifyDescriptor nd = new NotifyDescriptor.Confirmation("Do you want to ?",  
NotifyDescriptor.YES_NO_OPTION);
```

- Saisie:**

```
NotifyDescriptor nd = new NotifyDescriptor.InputLine("Value", "Default");
```



- Envoi du message et récupération du résultat:**

```
Object result = DialogDisplayer.getDefault().notify(nd);
```

GL Avancé: Prototypage et interfaces utilisateur

36 / 69

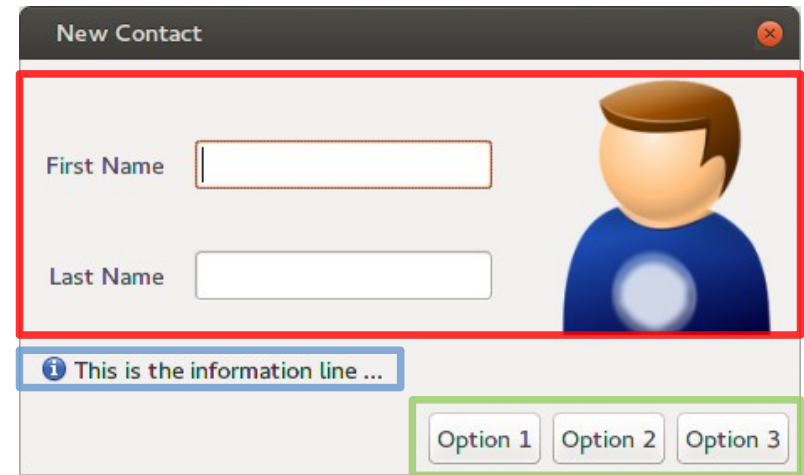
Boite de dialogue personnalisée

- La classe « `DialogDescriptor` » étend « `NotifyDescriptor` » :

```
MyPanel myPanel = new MyPanel();  
  
DialogDescriptor dd = new DialogDescriptor(  
    myPanel,  
    "New Contact",  
    true,  
    new String[]{"Option 1", "Option 2", "Option 3"},  
    "",  
    DialogDescriptor.DEFAULT_ALIGN,  
    HelpCtx.DEFAULT_HELP,  
    null);
```

```
//Notify users using info/warning/error messages in designed line at the bottom of your dialog  
dd.createNotificationLineSupport();  
dd.getNotificationLineSupport().setInformationMessage("This is the information line ...");
```

```
//Display the dialog  
Object returnValue = DialogDisplayer.getDefault().notify(dd);
```



GL Avancé: Prototypage et interfaces utilisateur

37 / 69

Boite de dialogue personnalisée

- **La gestion des erreurs nécessite la modification du `JPanel`**

- « `MyPanel` » doit implémenter l'interface « `DocumentListener` »
- Ajouter une constante pour le changement de propriété
- Redéfinir les méthodes « `insertUpdate` », « `removeUpdate` » et « `changedUpdate` »

```
public class MyPanel extends javax.swing.JPanel implements DocumentListener{

    public final static String PROP_FIRSTNAME = "firstName";

    public MyPanel() {
        initComponents();
        firstNameField.getDocument().addDocumentListener(this);
    }

    public void insertUpdate(DocumentEvent de) {
        if(firstNameField.getDocument() == de.getDocument())
            firePropertyChange(PROP_FIRSTNAME, false, true);
    }

    public void removeUpdate(DocumentEvent de) {...}
    public void changedUpdate(DocumentEvent de) {...}
}
```

GL Avancé: Prototypage et interfaces utilisateur

38 / 69

Boite de dialogue personnalisée

- **Gestion des erreurs : modifier le `DialogDescriptor`**

- Positionner le message d'erreur et invalider le bouton « OK »
- Enregistrer un listener de changement de propriété sur le panel

```
dd.getNotificationLineSupport().setErrorMessages("User name is required !");  
dd.setValid(false);
```

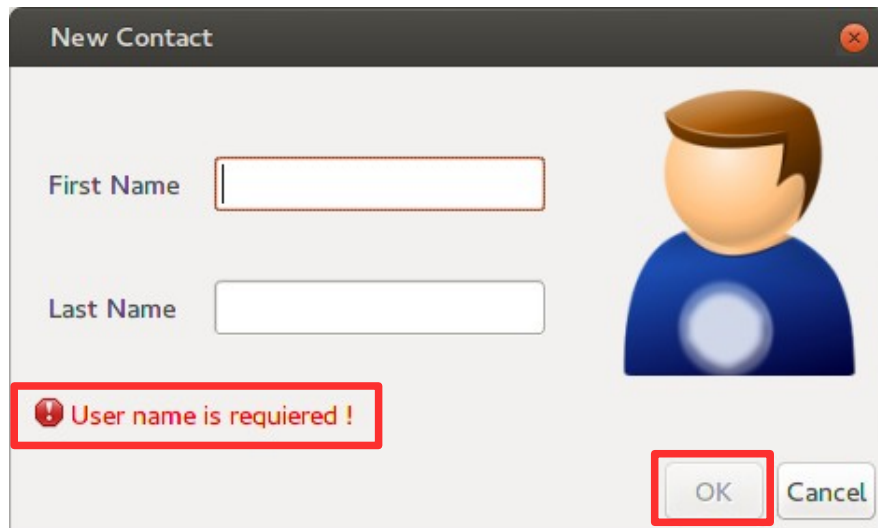
```
myPanel.addPropertyChangeListener(MyPanel.PROP_FIRSTNAME, new PropertyChangeListener()  
{  
    public void propertyChange(PropertyChangeEvent pce)  
    {  
        if(myPanel.getFirstName().trim().isEmpty())  
        {  
            dd.setValid(false);  
            dd.getNotificationLineSupport().setErrorMessages("First name is required !");  
        }  
        else  
        {  
            dd.setValid(true);  
            dd.getNotificationLineSupport().clearMessages();  
        }  
    }  
});
```

GL Avancé: Prototypage et interfaces utilisateur

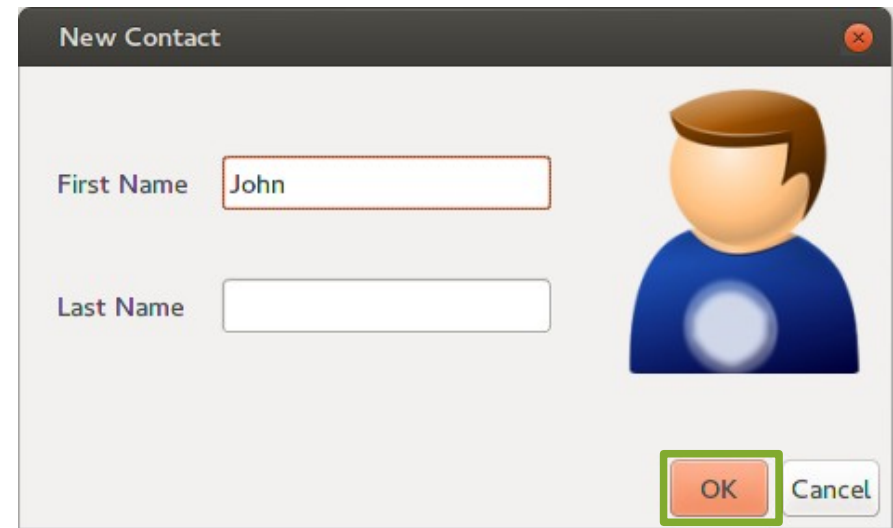
39 / 69

Boîte de dialogue personnalisée

- **Gestion des erreurs :**
 - Détection de la saisie de l'utilisateur
 - Mise à jour dynamique du message d'erreur
 - Validation du bouton « OK »



The dialog box titled "New Contact" contains two text input fields: "First Name" and "Last Name". The "First Name" field is empty and has a red border. Below the fields is a red-bordered box containing a red warning icon and the text "User name is required !". To the right of the fields is a stylized male user icon. At the bottom right, there are two buttons: "OK" and "Cancel". The "OK" button is disabled and has a red border, while the "Cancel" button is enabled and has a grey border.



The dialog box titled "New Contact" is shown in a second state. The "First Name" field now contains the text "John" and has a green border. The "Last Name" field remains empty. The red error message box is no longer present. The "OK" button is now enabled, highlighted with a green border, and has an orange background. The "Cancel" button remains disabled with a grey border.

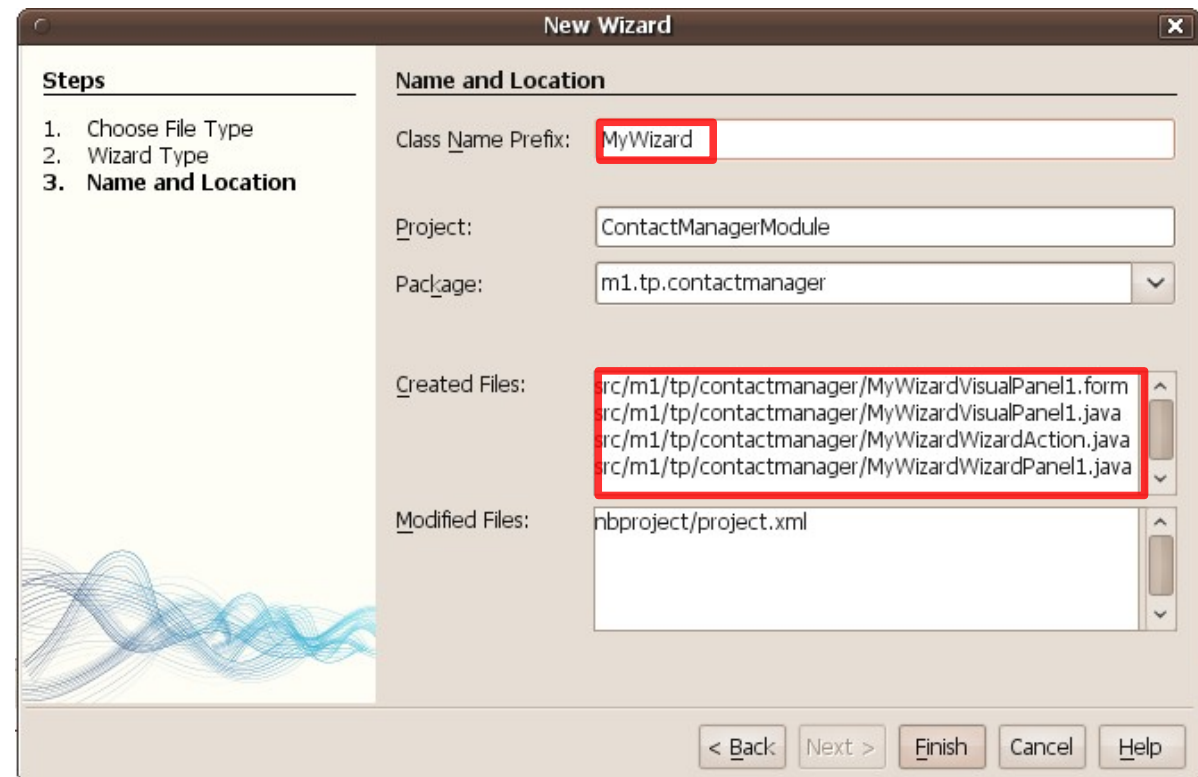
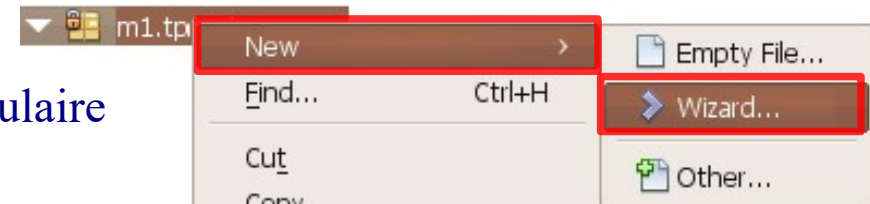
GL Avancé: Prototypage et interfaces utilisateur

40 / 69

Les assistants

- **Utiliser l'assistant « New Wizard » :**

- Code généré: Action, Wizard Descriptor et Formulaire
- Validation par la méthode « isValid() »



GL Avancé: Prototypage et interfaces utilisateur

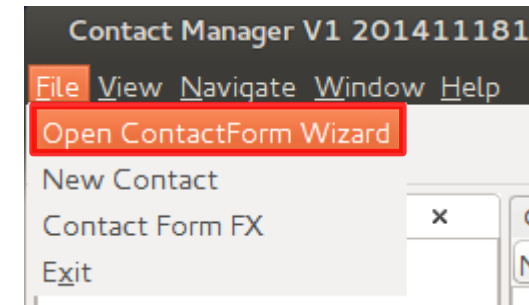
41 / 69

Les assistants

- **Classe «WizardAction» :**

- Dé-commenter les annotations pour activer l'action depuis le menu

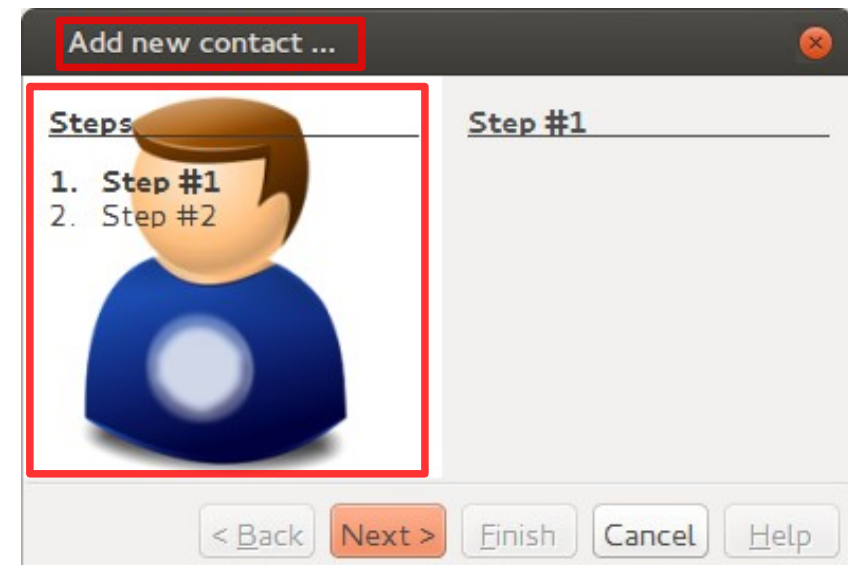
```
@ActionID(category="File", id="wizard.ContactFormWizardAction")
@ActionRegistration(displayName="Open Contact Form Wizard")
@ActionReference(path="Menu/File", position=1)
public final class ContactFormWizardAction implements ActionListener
{ [...] }
```



- Modifier l'aspect : WizardDescriptor.putProperty() ou Jcomponent.putClientProperty()

Ex : Ajouter un titre et une image dans la sidebar :

```
public void actionPerformed(ActionEvent e) {
    wiz.setTitle("Add new contact ...");
    wiz.putProperty(
        WizardDescriptor.PROP_IMAGE,
        ImageUtilities.loadImage("wizard/contact.png", true));
    [...]
}
```



GL Avancé: Prototypage et interfaces utilisateur

42 / 69

Les assistants

- **Classe «VisualPanel» :**

- Changer le nom de chaque étape :

```
public String getName()  
{  
    return "Identity";  
}
```

- « Form » pour l'interface utilisateur

- Créer les accesseurs pour tous les champs

```
public JTextField getFirstNamejTextField()  
{  
    return firstNamejTextField;  
}
```

```
public JTextField getLastNamejTextField()  
{  
    return lastNamejTextField;  
}
```

The screenshot shows a Java Swing dialog titled "Add new contact ...". It features a wizard interface with two steps: "1. Identity" and "2. Addresses". The "Identity" step is currently active and highlighted with a red box. To the left of the main form is a stylized user icon. The main form contains two text input fields labeled "First Name" and "Last Name". At the bottom of the dialog, there are five buttons: "< Back", "Next >", "Finish", "Cancel", and "Help". The "Next >" button is highlighted in orange, indicating the next step in the process.

GL Avancé: Prototypage et interfaces utilisateur

43 / 69

Les assistants

- **Classe «WizardPanel» :**

- Stocker les données saisies dans le store de l'instance du wizard :

```
public void storeSettings(WizardDescriptor wiz) {  
    // use wiz.putProperty to remember current panel state  
    wiz.putProperty("name", getComponent().getFirstNamejTextField().getText());  
    wiz.putProperty("address", getComponent().getLastNamejTextField().getText());  
}
```

- Lire les données à pré-charger depuis l'instancé du wizard :

```
public void readSettings(WizardDescriptor wiz) {  
    // use wiz.getProperty to retrieve previous panel state  
}
```

- Persistance globale en écriture (fichier wizard.properties) :

```
Preferences pref = NbPreferences.forModule(getClass());  
pref.put("namePreference", (String) wiz.getProperty("name"));
```

- Lire une donnée persisté globalement :

```
Preferences pref = NbPreferences.forModule(getClass());  
Pref.get("namePreference", "");
```

GL Avancé: Prototypage et interfaces utilisateur

Les assistants

- **Validation d'un écran : étendre l'interface** WizardDescriptor.ValidatingPanel

```
public class MyWizardPanel implements WizardDescriptor.ValidatingPanel<WizardDescriptor>{
    private boolean isValid = false;
    public void validate() throws WizardValidationException {
        isValid = false;
        String name = component.getFirstNameJTextField().getText();
        if(name.isEmpty())
            throw new WizardValidationException(null, "Invalid Name", null);
        isValid = true;
    }
    public boolean isValid() {
        return isValid;
    }
}
```

- **Classe « WizardAction » (suite) :**

- **Accès aux données saisies, depuis l'action :**

```
if (DialogDisplayer.getDefault().notify(wiz) == WizardDescriptor.FINISH_OPTION) {
    String name = (String) wiz.getProperty("name");
    String address = (String) wiz.getProperty("address");
    DialogDisplayer.getDefault().notify(
        new NotifyDescriptor.Message( "Name: "+name+"\nAddress: "+address));
}
```

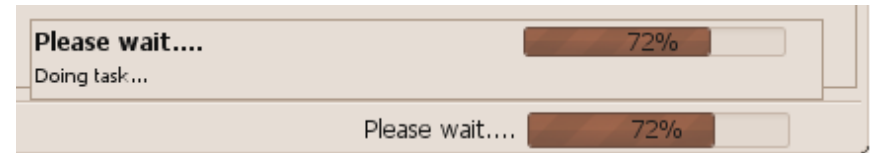


GL Avancé: Prototypage et interfaces utilisateur

45 / 69

Gestion de la progression

- Fourni par le module Progress API
- Utilisation du `ProgressHandleFactory`:



```
Runnable myRunnable = new Runnable()
{
    public void run()
    {
        ProgressHandle myProgressHandle = ProgressHandleFactory.createHandle("Please wait...");
        myProgressHandle.start(100);

        myProgressHandle.progress("Starting task...", 1);
        //DO TASK HERE
        myProgressHandle.progress("Ending task...", 100);

        myProgressHandle.finish();
    }
};
RequestProcessor.Task myTask = RequestProcessor.getDefault().post(myRunnable);
```

- Toujours lancer la tache dans un thread séparé pour ne pas bloquer le thread SWING !!

GL Avancé: Prototypage et interfaces utilisateur

46 / 69

Lookup global de sélection

- **Association d'un lookup de sélection avec un TopComponent**

- Code à appeler dans le constructeur du TopComponent :

```
InstanceContent contactSelectionContent = new InstanceContent() ;
associateLookup( new AbstractLookup(contactSelectionContent)) ;
```

- **Mise à jour de la sélection**

- Méthodes de la classe InstanceContent :

Modifier and Type	Method and Description
void	<code>add(Object inst)</code> The method to add instance to the lookup with.
<T,R> void	<code>add(T inst, InstanceContent.Convertor<T,R> conv)</code> Adds a convertible instance into the lookup.
void	<code>remove(Object inst)</code> Remove instance.
<T,R> void	<code>remove(T inst, InstanceContent.Convertor<T,R> conv)</code> Remove instance added with a convertor.
<T,R> void	<code>set(Collection<T> col, InstanceContent.Convertor<T,R> conv)</code> Changes all pairs in the lookup to new values.

- Exemple de définition d'une nouvelle sélection :

```
Collection<Contact> selectedContacts = Collections.singleton(new Contact());
contactSelectionContent.set(selectedContacts, null);
```

GL Avancé: Prototypage et interfaces utilisateur

47 / 69

Lookup global de sélection

- **Accès au lookup global de sélection des TopComponent**

```
Lookup global = Utilities.actionsGlobalContext() ;
```

- **Lookup résultant de la sélection d'un type d'objet**

```
Lookup.Result<Contact> lookupResult = global.lookupResult(Contact.class);
```

- **Écoute du changement de sélection**

```
lookupResult.addLookupListener( (LookupEvent le) -> checkSelection() ) ;
```

- **Récupération du résultat de la sélection courante**

- Méthodes de la classe `Lookup.Result` :

Modifier and Type	Method and Description
<code>abstract void</code>	<code>addLookupListener(LookupListener l)</code> Registers a listener that is invoked when there is a possible change in this result.
<code>Set<Class<? extends T>></code>	<code>allClasses()</code> Get all classes represented in the result.
<code>abstract Collection<? extends T></code>	<code>allInstances()</code> Get all instances in the result.
<code>Collection<? extends Lookup.Item<T>></code>	<code>allItems()</code> Get all registered items.
<code>abstract void</code>	<code>removeLookupListener(LookupListener l)</code> Unregisters a listener previously added.

- Exemple de code pour la méthode « `checkSelection` »:

```
Collection<? extends Contact> selectedContacts = lookupResult.allInstances() ;
```

GL Avancé: Prototypage et interfaces utilisateur

48 / 69

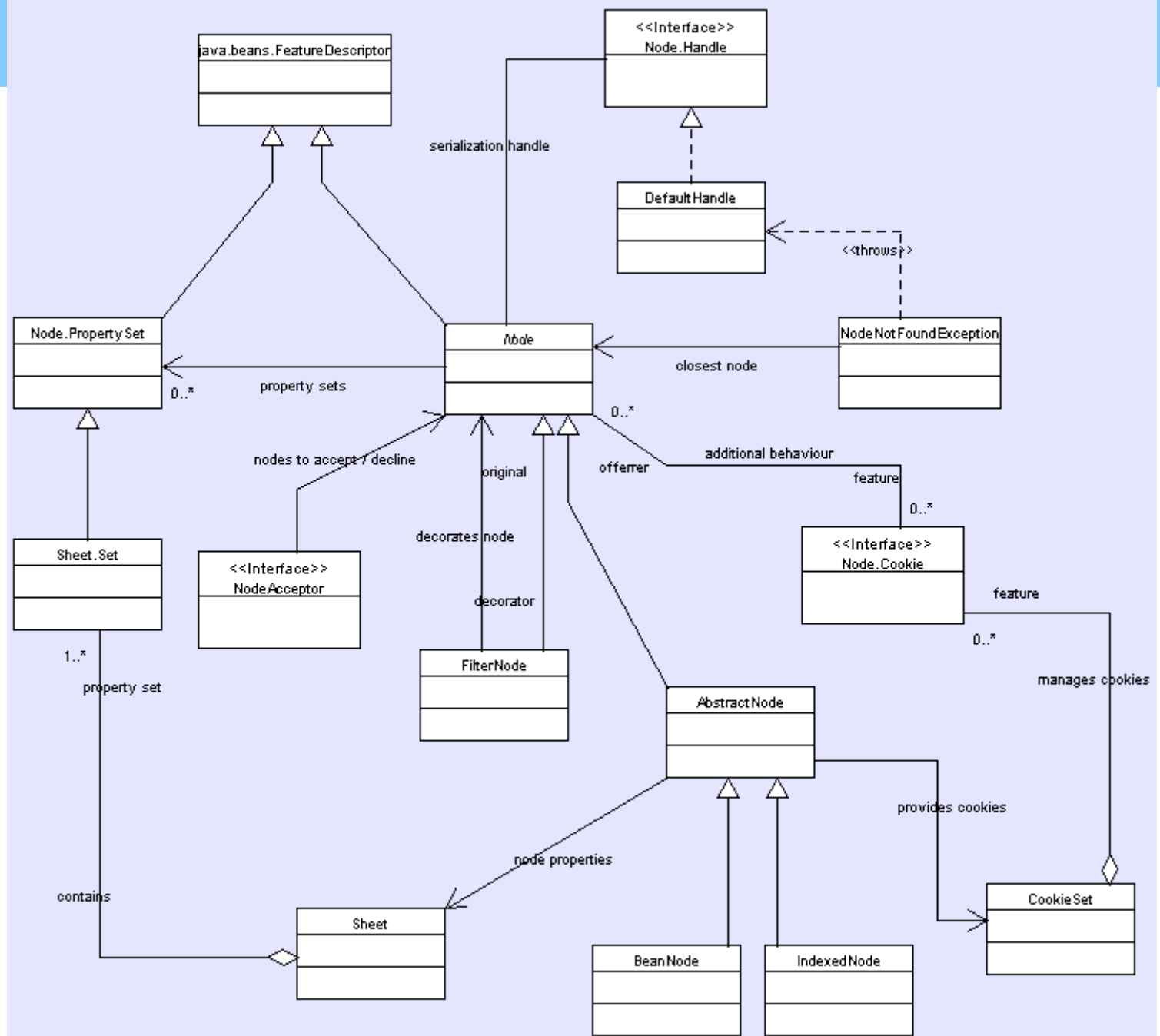
Nodes API

- **Définition**
 - Modèle générique de représentation et de visualisation des données
 - Paradigm Modèle – Vue – Controller :
 - ✓ Modèle : Root node, Child node
 - ✓ Vue : BeanTreeView, OutlineView
 - ✓ Controller : Root context
- **Personnalisation des icônes**
- **Gestion des actions sur les noeuds**
- **Mécanisme de sélection**
- **Filtrage par délégation**
- **Différentes vues et représentations possibles**
 - BeanTreeView
 - OutlineView
 - Master-Detail view
 - PropertySheetView
- **Fonctionnalité de recherche rapide intégrée**

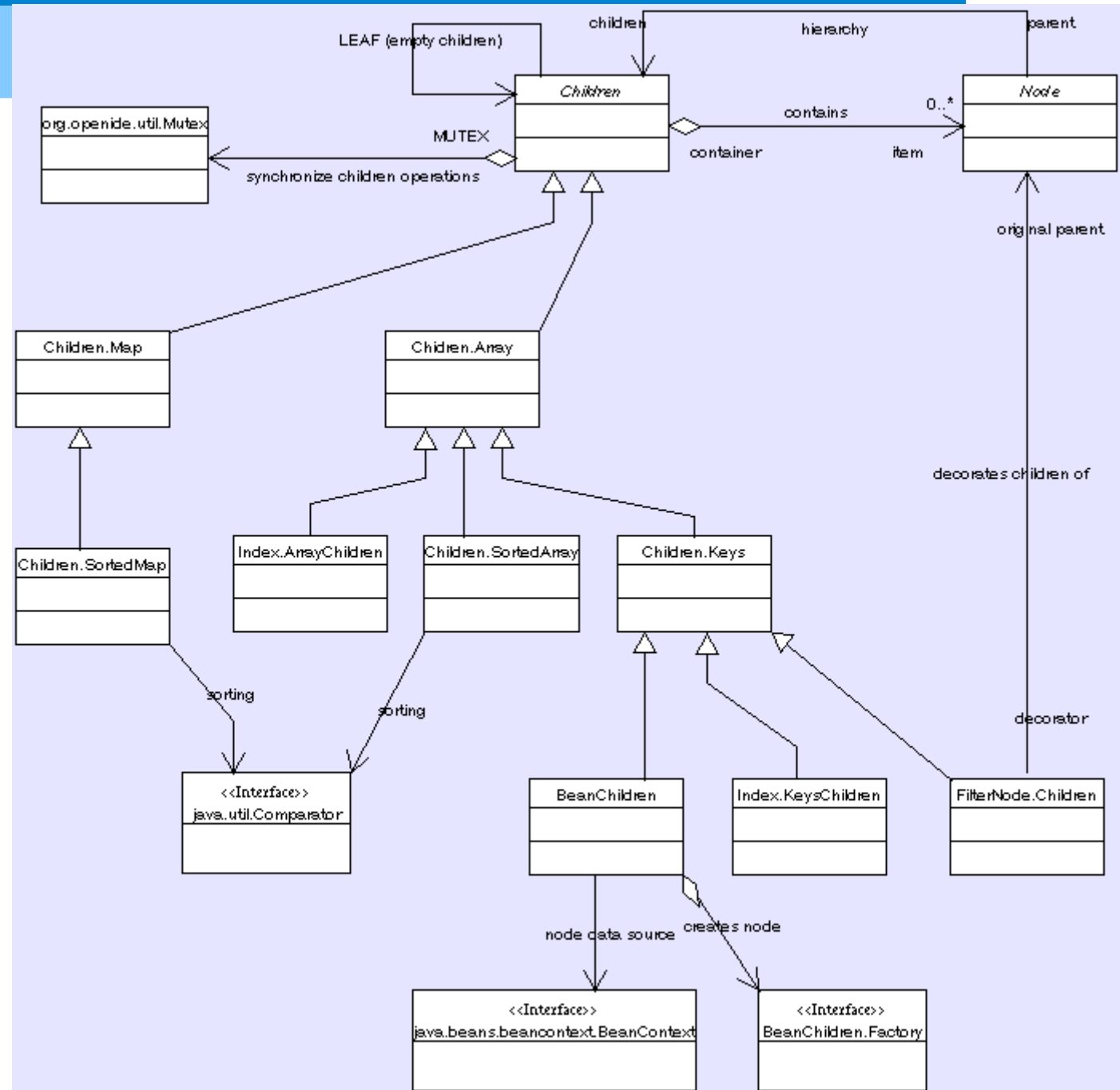
GL Avancé: Prototypage et interfaces utilisateur

Nodes API

- Structure



- **Hiérarchie**



GL Avancé: Prototypage et interfaces utilisateur

51 / 69

Nodes API : exemple

- **Noeud affichant un contact :**

```
public class ContactNode extends AbstractNode implements PropertyChangeListener {

    public ContactNode(Contact contact) {
        super(Children.LEAF, Lookups.singleton(contact)) ;
        setName(contact.getName()) ;
        setDisplayName(contact.getName()) ;
        setShortDescription(contact.getName()) ;
    }

    public String getHtmlDisplayName() {
        Contact contact = getLookup().lookup(Contact.class) ;
        return "<font color=red>" + contact.getName() + "</font>" ;
    }

    public void propertyChange(PropertyChangeEvent event) {
        fireDisplayNameChange(null, getDisplayName()) ;
    }

}
```

GL Avancé: Prototypage et interfaces utilisateur

52 / 69

Nodes API : exemple

- Factory de création de nœuds affichant un contact :**

```
public class ContactChildFactory extends ChildFactory<Contact> {  
  
    private final ContactManager contactManager ; // provides available contacts  
  
    public ContactChildFactory() {  
        contactManager = Lookup.getDefault().lookup(ContactManager.class) ;  
        contactManager.addPropertyChangeListener(event → refresh(true)) ;  
    }  
  
    protected boolean createKeys(List<Contact> contacts) {  
        contacts.addAll(contactManager.getContacts()) ;  
        Return true ;  
    }  
  
    protected Node createNodeForKey(Contact contact) {  
        ContactNode node = new ContactNode(contact) ;  
        contact.addPropertyChangeListener(WeakListeners.propertyChange(node, contact)) ;  
        return node ;  
    }  
}
```


GL Avancé: Prototypage et interfaces utilisateur

53 / 69

Nodes API : exemple

- **Noeud racine :**

```
public class RootNode extends AbstractNode {  
  
    public RootNode() {  
        super(Children.create(new ContactChildFactory(), false)) ;  
        SetDisplayName("All contacts") ;  
        setShortDescription("This is the root node") ;  
    }  
}
```

- **Affichage de l'arborescence :**

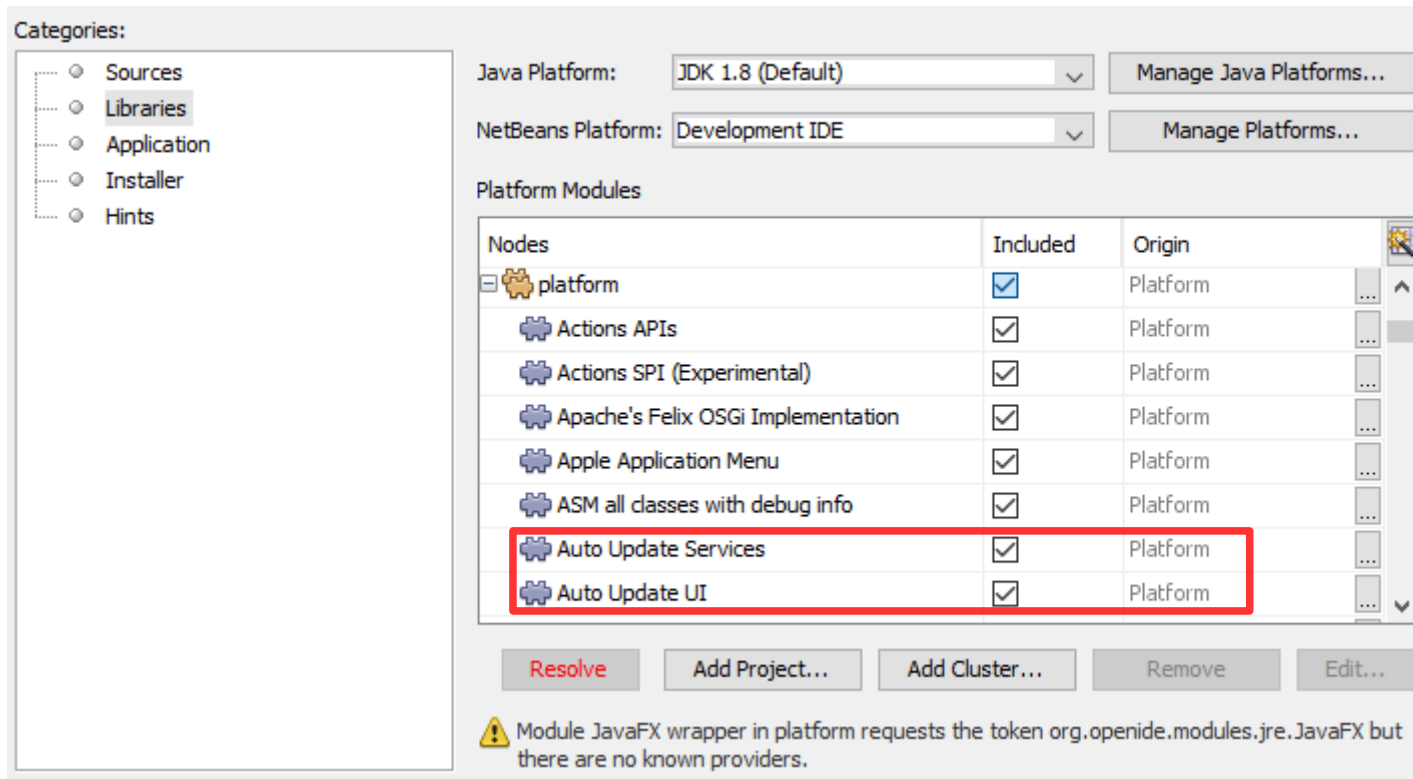
```
public class ContactTopComponent extends TopComponent implements ExplorerManager.Provider {  
    private final ExplorerManager explorer = new ExplorerManager() ;  
  
    public ContactTopComponent() {  
        initComponents() ;  
        [...]  
        BeanTreeView view = new BeanTreeView() ;  
        add(view, BorderLayout.CENTER) ;  
        associateLookup(ExplorerUtils .createLookup(explorer, getActionMap())) ;  
        Explorer.setRootContext(new RootNode()) ;  
    }  
  
    public ExplorerManager getExplorerManager() {  
        return explorer;  
    }  
}
```

GL Avancé: Prototypage et interfaces utilisateur

54 / 69

Mise à jour de l'application

- **Auto Update Service :**
 - Service de mise à jour automatique des modules
 - Activation du service :
 - ✓ Dépendance vers le module "Auto Update Service"
 - ✓ Dépendance vers le module "Auto Update UI"



GL Avancé: Prototypage et interfaces utilisateur

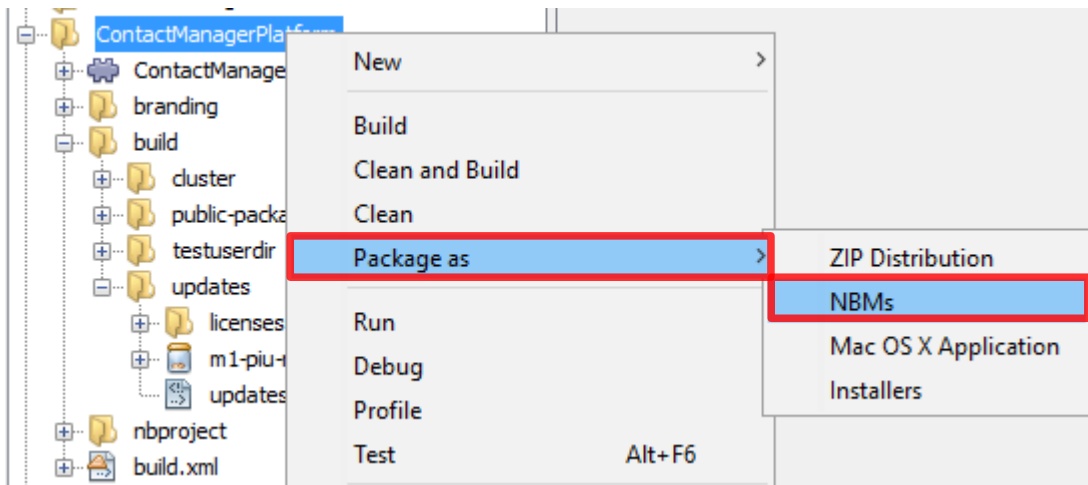
55 / 69

Mise à jour de l'application

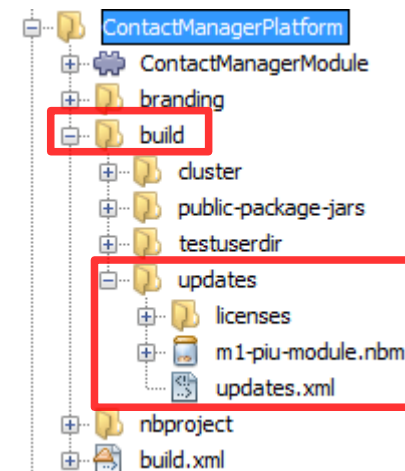
- **Auto Update Service :**

- Création du centre de mise à jour :

- ✓ URL pointant vers un fichier "updates.xml" et la liste des modules dans leur dernière version
- ✓ Automatisation de la création :
 - Menu contextuel depuis un projet "Package as → NBMs"



- Génération des fichiers dans "build / updates"



GL Avancé: Prototypage et interfaces utilisateur

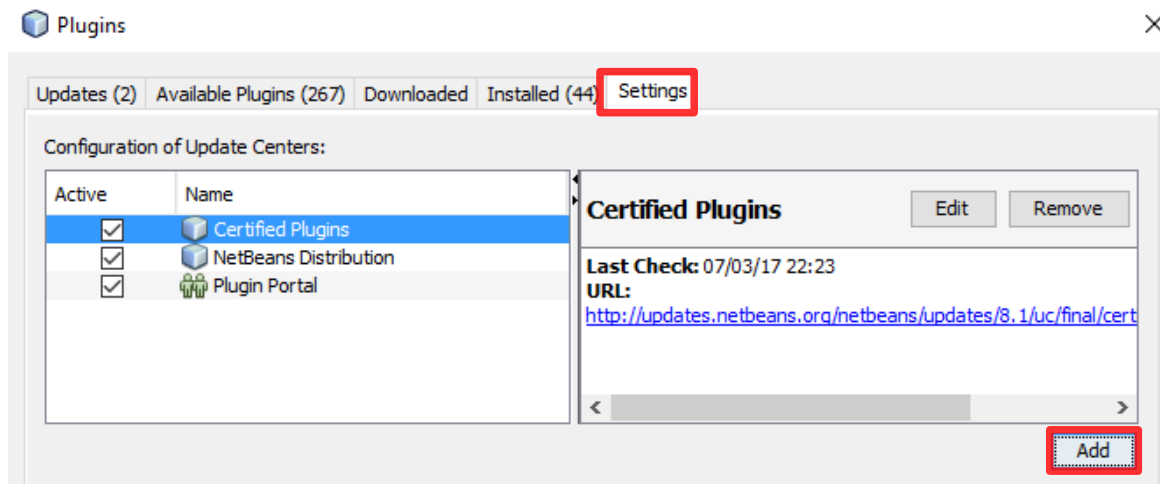
56 / 69

Mise à jour de l'application

- **Auto Update Service :**

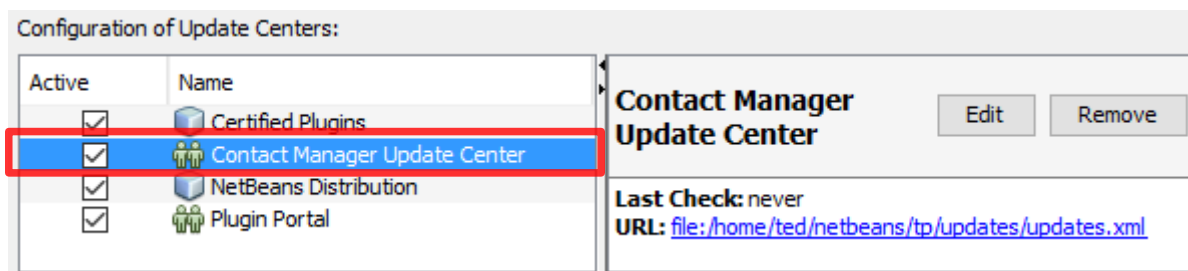
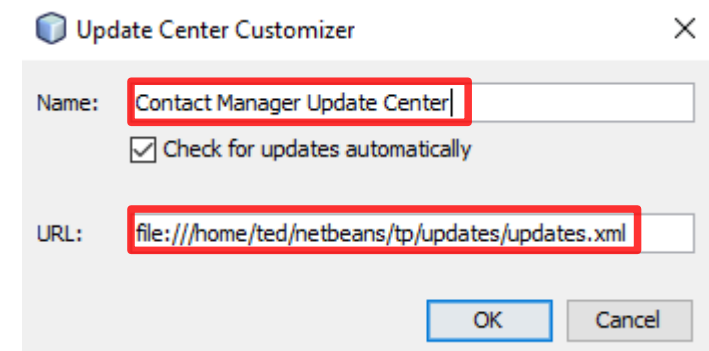
- Configuration de la mise à jour

- ✓ Plugin manager : menu "Tools → Plugins", puis bouton "Add" pour ajouter le centre d'update



- ✓ Update center customizer : nom et URL (http ou fichier local)

- ✓ Le centre d'update est configuré et apparaît dans la liste



GL Avancé: Prototypage et interfaces utilisateur

57 / 69

JavaFX intégration

- **Intégration dans NetBeans**

- Dépendance de l'application « NetBeans Platform »:
 - ✓ JDK 1.8
 - ✓ ou JDK 1.7 + wrapper « jfxrt.jar » dans un module
- Inclure le fichier FXML dans un module (éventuellement avec le contrôleur Java, le CSS)
- Utiliser **JXPanel** pour faire cohabiter JavaFX avec Swing (dans un `TopComponent` par ex.)

```
private void initFX() {  
    fxPanel = new JFXPanel();  
    setLayout(new BorderLayout());  
    add(fxPanel, BorderLayout.CENTER);  
  
    Platform.setImplicitExit(false); //Ensure JavaFX thread does not close application  
  
    Platform.runLater(()-> { //Create FX scene outside Swing Thread  
        createFXScene();  
    });  
}
```

GL Avancé: Prototypage et interfaces utilisateur

58 / 69

JavaFX intégration

- **Intégration dans NetBeans**

```
private void createFXScene() {  
    try {  
        Parent root = FXMLLoader.load(getClass().getResource("ContactPanel.fxml"));  
        Scene scene = new Scene(root, Color.LIGHTGREY);  
        fxPanel.setScene(scene);  
    } catch (IOException e) {  
        Exceptions.printStackTrace(e);  
    }  
}
```

- **Attention aux manipulations JavaFX et Swing :**

- Utiliser `SwingUtilities.invokeLater()` pour modifier un Swing depuis JavaFX
- Utiliser `Platform.runLater()` pour modifier un JavaFX depuis Swing

- **Le composant inverse pour insérer du Swing dans JavaFX : `SwingNode`**

GL Avancé: Prototypage et interfaces utilisateur

59 / 69

Internationalisation

- La gestion de l'I18N est automatisée
- Utilisation d'un ressource bundle
 - Un fichier « `Bundle.properties` » par paquetage :

```
TopComponentPanel.jLabel1.text=Nom:  
TopComponentPanel.jLabel2.text=Prénom:
```

- Appel dans le code source :

```
Mnemonics.setLocalizedText(  
    jLabel1,  
    NbBundle.getMessage(TopComponentPanel.class, "TopComponentPanel.jLabel1.text"));
```

- Internationalisation des images (non automatisé)

```
Image image = ImageUtilities.loadImage("resources/icon.png", true);
```

- Ajouter une langue

- Créer un fichier « `Bundle_<language>_<country>.properties` » dans le même répertoire que le fichier « `Bundle.properties` »

GL Avancé: Prototypage et interfaces utilisateur

60 / 69

Internationalisation

- **Ressource d'un module**

- Utilisation du protocole « nbresloc »

```
URL url = new URL("nbresloc:/m1/pi/contactmanager/resources/icon.png");  
ImageIcon icon = new ImageIcon(url);
```

- **Internationalisation des folders du layer.xml**

- Utilisation des attributs « `SystemFileSystem.localizingBundle` » et
« `SystemFileSystem.icon` »

```
<folder name="Menu">  
  <attr name="SystemFileSystem.localizingBundle" stringvalue="m1.pi.contactmanager.Bundle"/>  
  <attr name="SystemFileSystem.icon" urlvalue="nbresloc:/m1/pi/contactmanager/resources/icon.png"/>  
</folder>
```

- **Localisation d'un module**

- Séparer les fichiers localisés dans un module portant l'extension de la langue et du pays
- Placer toutes les traductions dans le répertoire « `locale` » du module à traduire

GL Avancé: Prototypage et interfaces utilisateur

61 / 69

Internationalisation

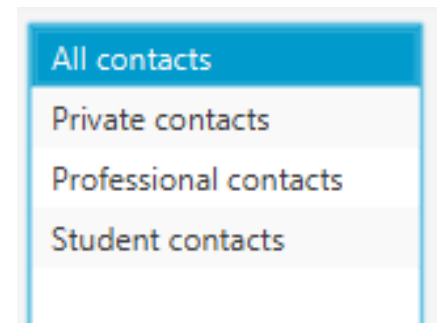
- **Internalisation pour JavaFX**

- Définition du ResourceBundle :

```
private void createFXScene() {  
    ResourceBundle resources = ResourceBundle.getBundle("my.app", Locale.getDefault());  
  
    try {  
        FXMLLoader fxmlloader = new FXMLLoader() ;  
        fxmlloader.setResources(resources) ;  
        Parent root = fxmlloader.load(getClass().getResource("ContactPanel.fxml"));  
        Scene scene = new Scene(root, Color.LIGHTGREY);  
        fxPanel.setScene(scene);  
    } catch (IOException e) {  
        Exceptions.printStackTrace(e);  
    }  
}
```

- Utilisation des clés d'internationalisation en FXML avec la notation %key :

```
<ListView fx:id="addressBooksListView" ...>  
    <items>  
        <FXCollections fx:factory="observableArrayList">  
            <String fx:value="%allContactsKey"/>  
            [...]   
        </FXCollections>  
    </items>  
</ListView>
```



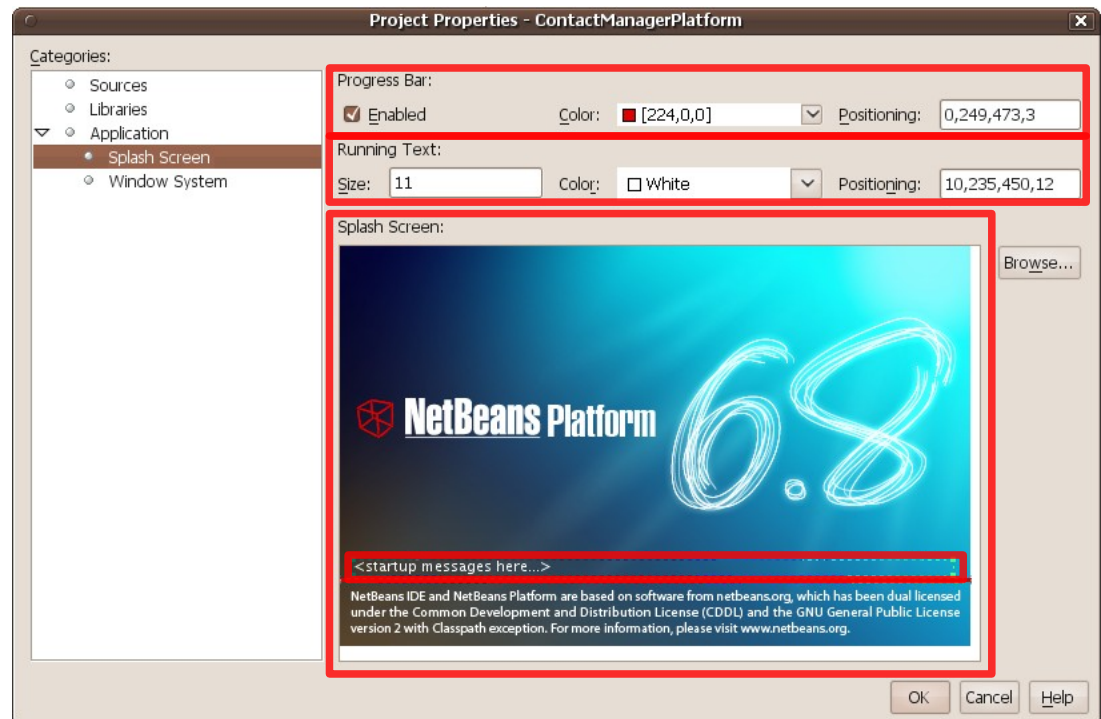
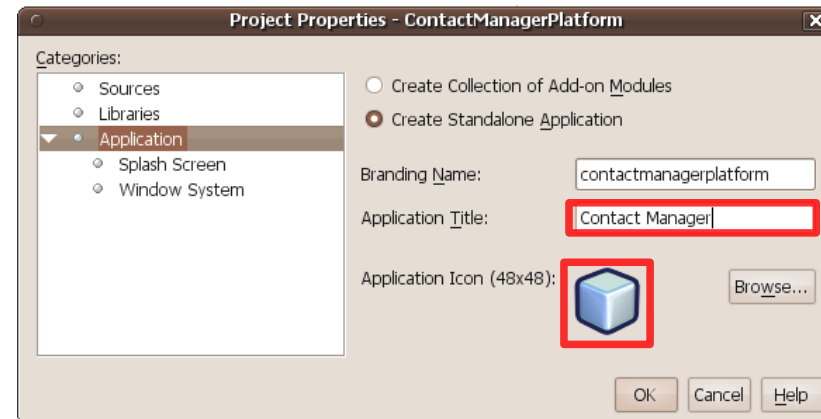
GL Avancé: Prototypage et interfaces utilisateur

62 / 69

Branding de l'application

- **Personnalisation de la maquette**

- Nommer la fenêtre
- Associer les icônes
- Définir un écran de démarrage
- Barre de progression
- Messages de chargement
- Taille de la police



GL Avancé: Prototypage et interfaces utilisateur

63 / 69

Branding de l'application

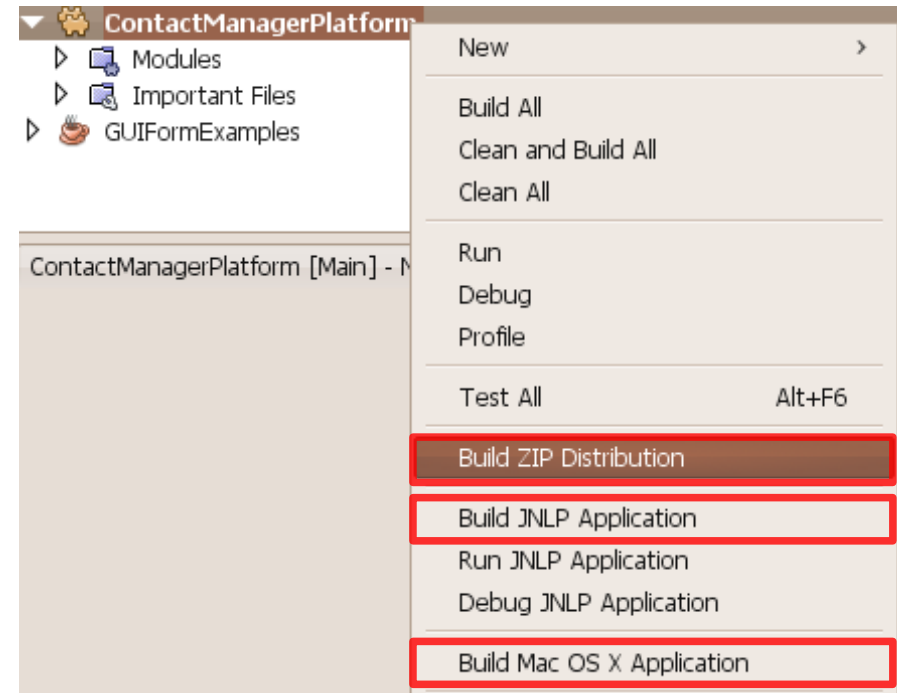
- **Système de fenêtrage : activation/désactivation de fonctionnalités**
 - **Window drag & drop** : possibilité de réorganisation le layout des fenêtres en glissant les fenêtres vers de nouvelles positions
 - **Floating windows** : possibilité de détacher les fenêtres dans un cadre indépendant
 - **Sliding windows** : possibilité de minimiser une fenêtre dans la barre latérale
 - **Maximized windows** : possibilité de maximiser une fenêtre en cliquant sur son entête
 - **Closing windows** : possibilité de fermer une fenêtre de document/non-document
 - **Window resizing** : possibilité d'ajuster la longueur et la largeur des fenêtres en déplaçant les barres de séparation
 - **Respect minimum size when resizing windows** : possibilité de redimensionner les fenêtres internes vers une taille 0 en utilisant les barres de séparation

GL Avancé: Prototypage et interfaces utilisateur

64 / 69

Distribution

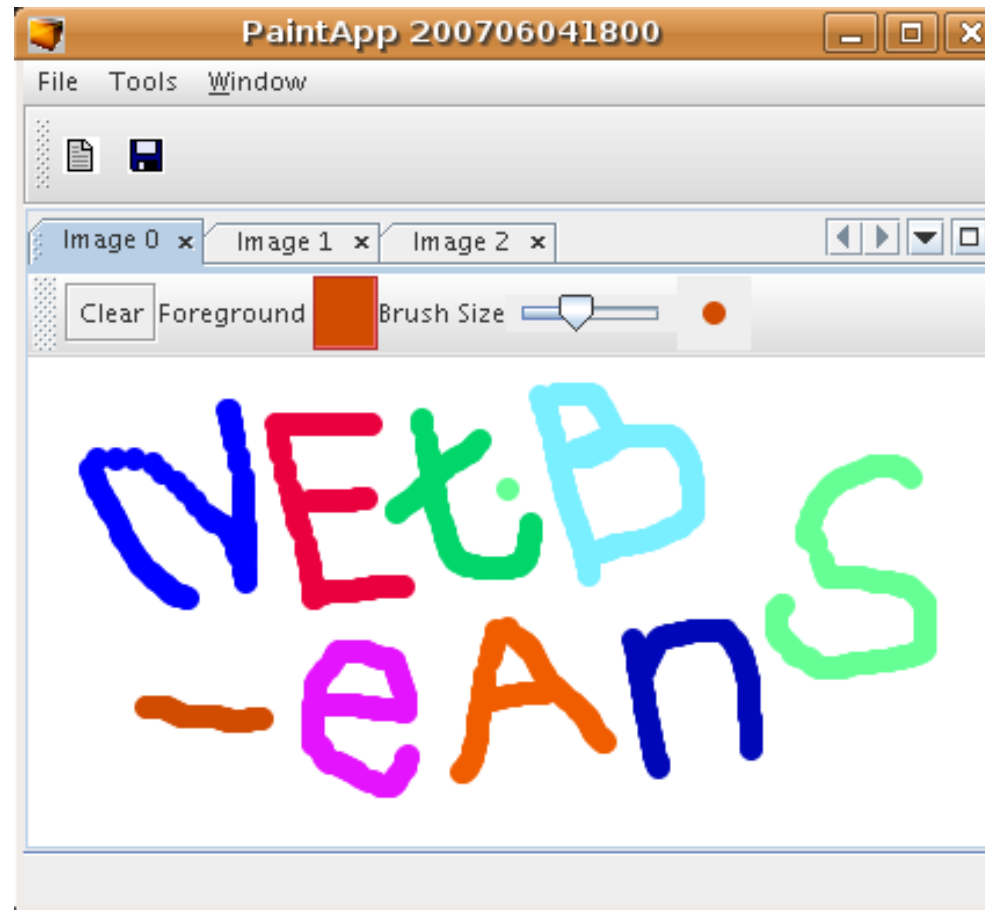
- **NetBeans est multi-plateforme:**
 - Création d'une distribution ZIP
 - ✓ Exécutable Windows
 - ✓ Script Shell Linux
 - Création d'une application Mac OS X
 - Création d'une application Java Web Start
 - Création d'un « installer »
 - ✓ Windows, Linux, MacOS, Solaris
 - ✓ Choix de la licence (aucune, GPL v2, ...)
 - ✓ Compression pack200 (diminue la taille de 70%)

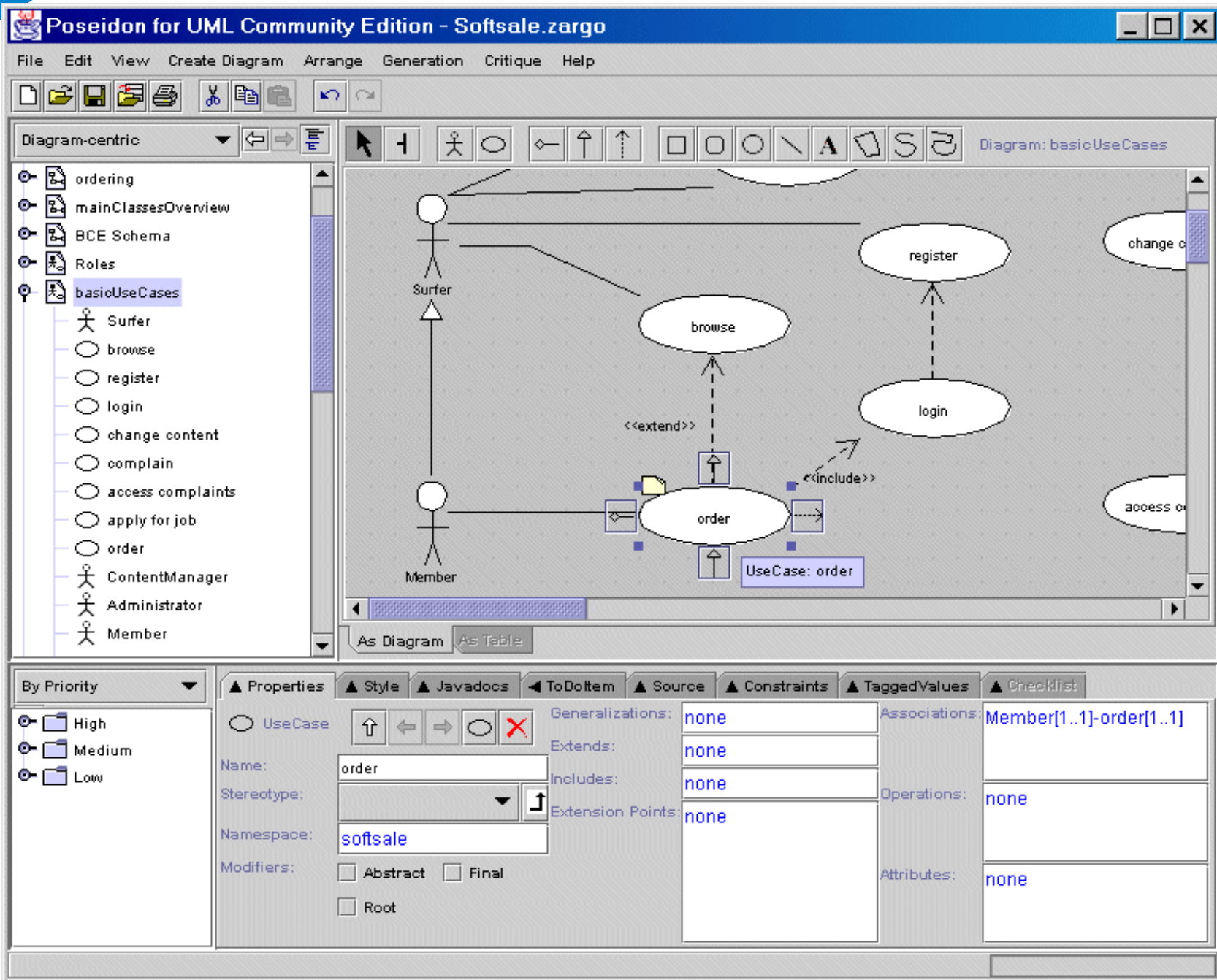


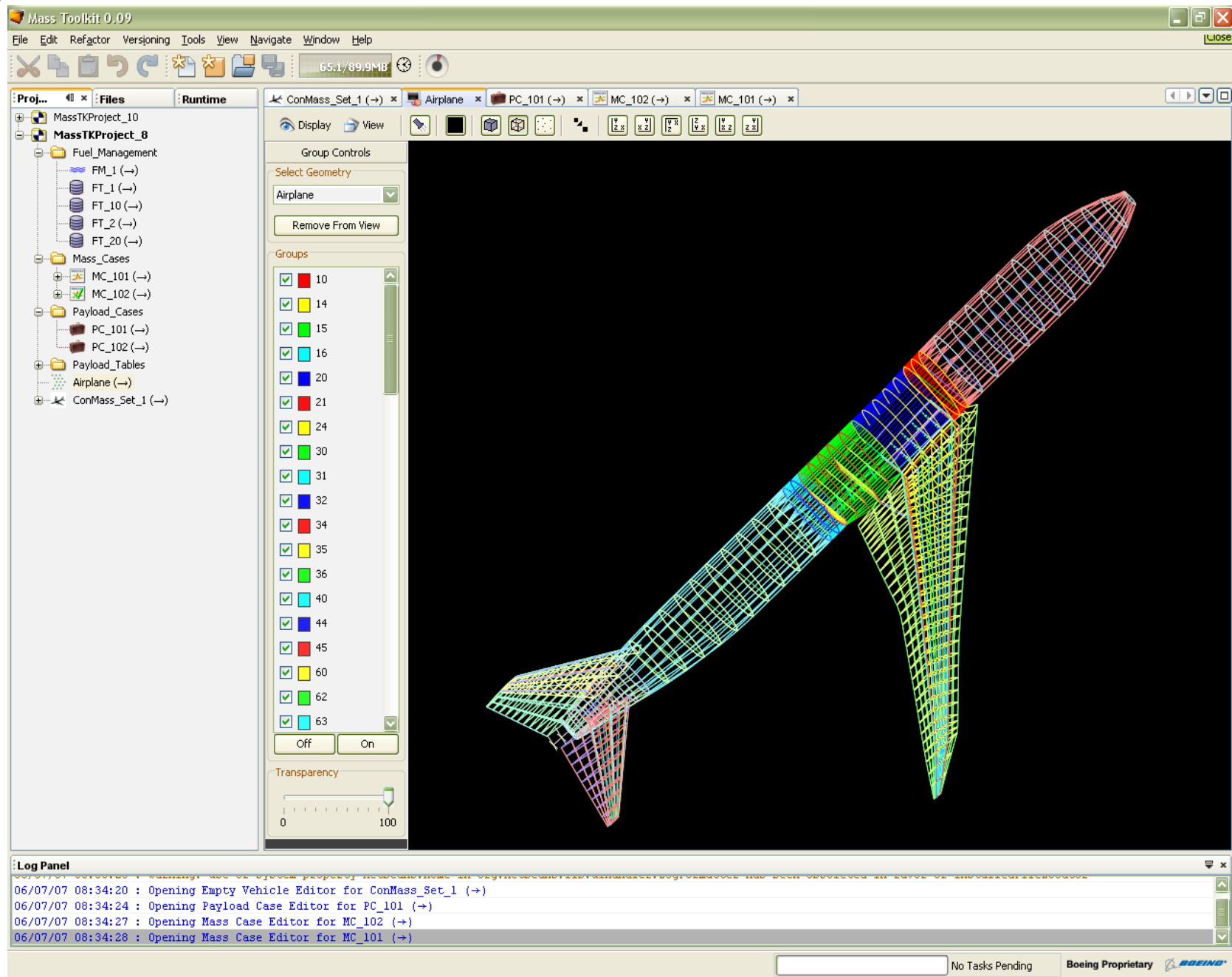
GL Avancé: Prototypage et interfaces utilisateur

65 / 69

Réalisations basées sur NetBeans Platform







iReport 3.5.0

File Edit Format View Preview Window Tools Help

Empty datasource

Repository Navigator

- All Charts Aegean Report
- All Charts Aegean Report
- Input controls
- Resources
 - allCharts.properties_label
 - allCharts_ro.properties_label
- All Charts Eye Candy Report
- All Charts Report
- Employee Accounts
- Employee List

Report Inspector

- SalesByMonth
 - Styles
 - Parameters
 - Fields
 - Variables
 - Scriptlets
 - Background
 - Title
 - [0, 1, 515, 1]
 - *repo:/images/1...
 - \$R{title}
 - [160, 40, 300, 30]
 - \$R{param.checkbox}
 - [160, 100, 300, 30]
 - \$R{param.list}
 - \$P{listInput}
 - Page Header
 - Column Header
 - Detail
 - [0, 0, 515, 15]
 - *repo:/SalesByMo...
 - Pie 3D Chart
 - Line Chart
 - Column Footer
 - Page Footer
 - Last Page Footer

Designer

XML Preview

Arial 22

\$R{title}

\$R{param.checkbox}

\$R{param.list}

\$P{TextInput}

\$P{ListInput}

\$F{close_month_name} + " " + \$F{close_month_name} + " " + \$F{close_month_name}

Detail

Page Header

Page Footer

Page + " " + String.

jasperreports

\$P{REPORT_SCRIPTLET}.message()

Palette

- Report Elements
 - Break
 - Crosstab
 - Frame
 - Line
 - Round Rectangle
 - Subreport
 - Chart
 - Ellipse
 - Image
 - Rectangle
 - Static Text
 - Text Field
- Tools

Formatting Tools Window

\$R{title} - Properties

Properties

Left	160
Top	10
Width	355
Height	30
Forecolor	[0, 0, 0]
Backcolor	[255, 255, 255]
Opaque	<input type="checkbox"/>
Style	Title
Key	
Position Type	Fix Relative to Top
Stretch Type	No stretch
Print Repeated Values	<input checked="" type="checkbox"/>
Remove Line When Blank	<input type="checkbox"/>

\$R{title}

Styles Library Window

- Header 1
- Header 2
- Header 3

Report Problems Window

Description	Object
java.lang.ClassNotFoundException: test.TestScriptlet	Arial_Normal
java.lang.ClassNotFoundException: com.jaspersoft.jasperserver.api.metadata.user.domain.User	LoggedInUser

iReport output

GL Avancé: Prototypage et interfaces utilisateur

69 / 69

Bibliographie

- "Rich Client Programming: Plugging into the NetBeans Platform", Tim Boudreau
- "The Definitive Guide to NetBeans Platform", Heiko Bock
- "100 NetBeans IDE Tips and Tricks", Ruth Kusterer
- "JavaFX Rich Client Programming on the NetBeans Platform ", Addison-Wesley
- <http://netbeans.org/kb/articles/books.html>
- <http://platform.netbeans.org/>
- <http://bits.netbeans.org/dev/javadoc/>