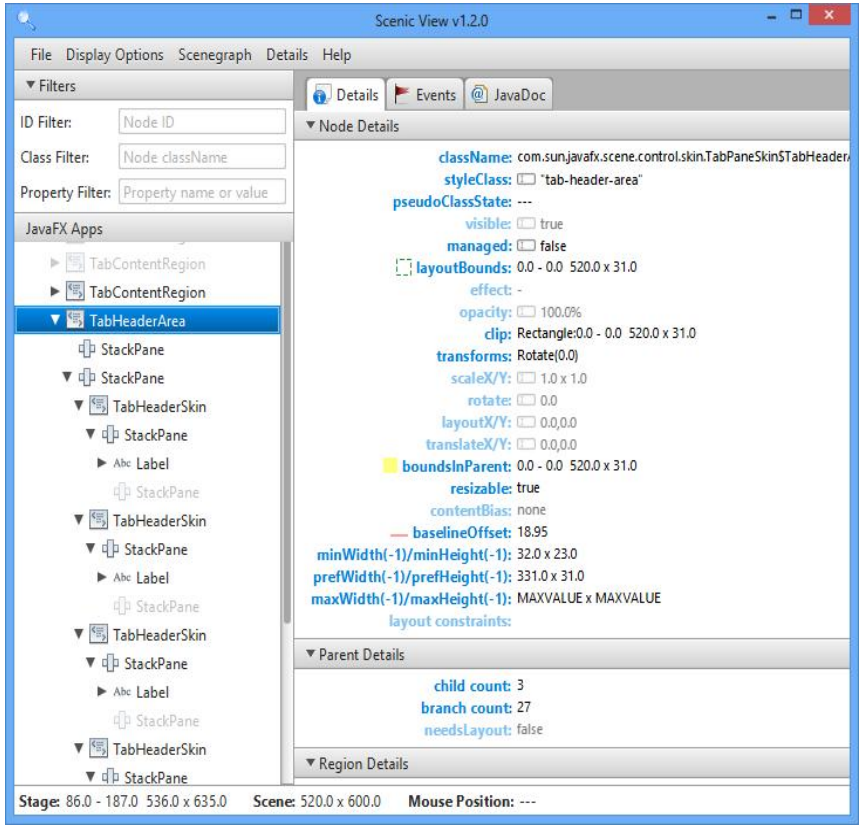
Tests et outils

LES OUTILS DE BASE

SceneBuilder et ScenicView

- SceneBuilder est l'outil officiel pour la construction de FXML de manière visuelle
- ScenicView est un outil externe (fxexperience.com)
  - Inspection d'une application JavaFX en live
  - Modifications de propriétés en live



Le composant WebView

Debug HTML5 dans JavaFX

- Utilisation de FireBug intégrable dans JavaFX
- Injection via le WebEngine

```
engine.executeScript("code standard d'injection FireBug Lite");
```

Les IDE disponibles

E(fx)CLIPSE

Forces (interne) : basé sur Eclipse, gratuit et open source, adoption très large, support de JavaFX efficace, support de FXGraph

Opportunités (externe) : développement de plugins, support de nombreuses technologies et langages, intégration définitive du plugin dans Eclipse

Faiblesses (interne) : performances, plugins de qualités inégales, intégration Maven et WTP instables, nécessite souvent l'ajout de plugins

Menaces (externe) : support communautaire pas toujours adapté aux entreprises, performances pouvant se dégrader, instabilité dans certaines versions

Les IDE disponibles

Intellij

Forces (interne) : Performances, robustesse, intégration Maven, support JavaFX efficace, haut niveau de personnalisation

Opportunités (externe) : Support de nombreuses technologies et langages, support de Java 8 et lambdas, inspections de code très étendues

Faiblesses (interne) : Payant (500$ pour une licence professionnelle), moins répandu en entreprise qu'Eclipse à ce jour

Menaces (externe) : Evolution du support JavaFX dépendant du succès de la technologie

Les IDE disponibles

NetBeans

Forces (interne) : Supporté par Oracle (solution officielle), intégration de JavaFX bien documentée

Opportunités (externe) : Support de JavaFX assuré par Oracle dans les prochaines versions

Faiblesses (interne) : Adoption marginale, intégration à des technologies non Oracle limitée, écosystème peu étendu

Menaces (externe) : Adoption de NetBeans demeurant marginale

Tests et IHM

Plusieurs options

- Plusieurs solutions existent
  - Configuration manuelle de tests spécifique pour JavaFX
  - Utiliser le toolkit JemmyFX d'Oracle
  - Le framework Open Source MarvinFX
  - La solution payante de SmartBear TestComplete
- Tester l'IHM est
  - Coûteux
  - Instable de temps à autres
  - Difficilement maintenable
  - Complexe et long à mettre en place
  - Plus fiables que des tests manuels car automatiques
  - A terme moins coûteux et couvrant un plus grand scope que des tests manuels

JemmyFX

L'outil officiel

- Disponible sur le repository de JavaFX (mercurial)
  - Build manuel du projet
  - Déploiement de l'artifact manuel dans Maven
  - L'intégration dans une usine logicielle n'est donc pas native
- Solution officielle utilisée par Oracle pour les tests IHM
  - API bas-niveau
  - Utilisée pour tester chaque composant de JavaFX
  - Ajoute des abstractions nommées « Dock » et « Wrap » pour effectuer des tests
  - Il faut étendre une classe de test JemmyFX
  - Utilise par défaut un mode d'émulation par événement JavaFX mais peut utiliser un mode par robot (événement de type système)

JemmyFX

Exemple

```
public class ButtonsSample extends SampleBase {

    private static SceneDock scene;
    private static LabeledDock status;

    @BeforeClass
    public static void startApp() throws InterruptedException {
        startApp(ButtonsApp.class);
        scene = new SceneDock();
        status = new LabeledDock(scene.asParent(), "status");
    }

    // First thing first - how to find a button.
    @Test
    public void lookup() {
        //a button is a labeled, every labeled could be found by text
        assertEquals(Button.class, new LabeledDock(scene.asParent(), "button",
                StringComparePolicy.EXACT).wrap().getControl().getClass());
        assertEquals(CheckBox.class, new LabeledDock(scene.asParent(), "tri-state",
                StringComparePolicy.EXACT).wrap().getControl().getClass());
    }

    // Push is, simply, a click.
    @Test
    public void push() {
        new LabeledDock(scene.asParent(), "button", StringComparePolicy.EXACT).mouse().click();
        //now check that the status line is updated
        status.wrap().waitProperty(Wrap.TEXT_PROP_NAME, "button pushed");
    }
}
```

TestFX

La solution de SmartBear

- Originellement intégré dans LoadUI (produit payant)
  - Intégré dans la solution payante TestComplete
  - Peut être utilisé seul en open source
  - https://github.com/SmartBear/TestFX
- SmartBear est un éditeur connu surtout pour son produit SoapUI

TestComplete

LoadUI

TestFX

Exemple

```
@Category( TestFX.class )
public class LabelForTest extends GuiTest {

      @Override
      protected Parent getRootNode() {
            TextField uText = TextFieldBuilder.create().id( "uname" ).build();
            TextField pText = PasswordFieldBuilder.create().id( "pword" ).build();
            Label u = LabelBuilder.create().text( "User name" ).labelFor(uText).build();
            Label p = LabelBuilder.create().text( "Password" ).labelFor( pText ).build();

            GridPane grid = new GridPane();
            grid.add( u, 0, 0 );
            grid.add( uText, 1, 0 );
            grid.add( p, 0, 1 );
            grid.add( pText, 1, 1 );
            return grid;
      }

      @Test
      public void shouldClickButton() {
            click( nodeLabeledBy("User name") ).type( "Steve" );
            click( nodeLabeledBy("Password") ).type( "duke4ever" );
            verifyThat( "#pword", hasText( "duke4ever" ) );
      }
}
```

MarvinFX

Alternative open source

- Solution open source de test pour JavaFX
  - https://github.com/guigarage/MarvinFX

- Micro framework de test pour JavaFX
- Simple d'utilisation
- Peu de features
- Efficace

MarvinFX

Exemple

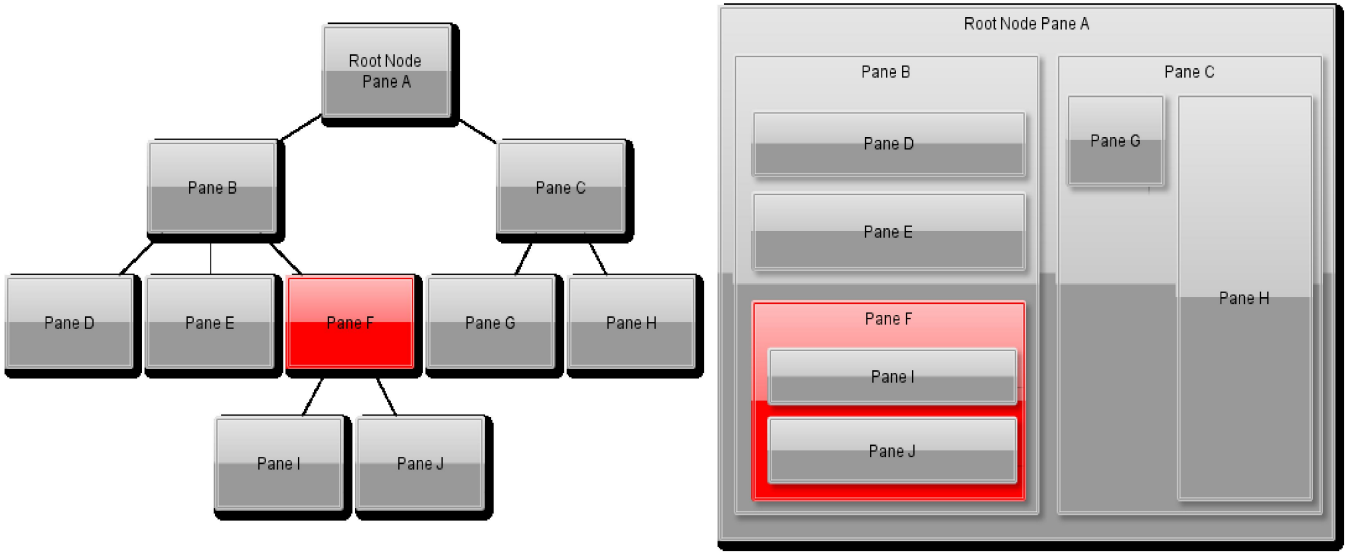
```
public class UnitTests {
	@Test
	public void test1() {
		final Button b1 = new Button("Hello MarvFX");
		BorderPane pane = new BorderPane();
		pane.setCenter(b1);
		MarvinFx.show(pane);
		MarvinFx.sleep(2000);
		NodeFixture<Button> b1Fixture = new NodeFixture<Button>(b1);
		b1Fixture.mouse().click();
		MarvinFx.sleep(2000);
	}

	@Test
	public void test2() {
		final Button b1 = new Button("Hello MarvFX 2");
		Stage s = MarvinFx.show(b1);
		s.setX(0);
		s.setY(0);
		MarvinFx.sleep(2000);
		NodeFixture<Button> b1Fixture = new NodeFixture<Button>(b1);
		b1Fixture.mouse().click();
		MarvinFx.sleep(2000);
	}
}
```

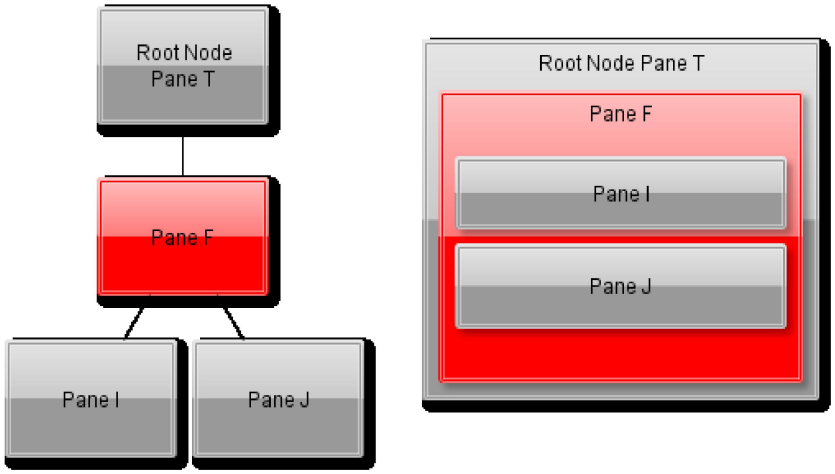
Testabilité d'une IHM

Comment fa

- Contexte applic



- Contexte de test



COMPOSANTS TESTABLES

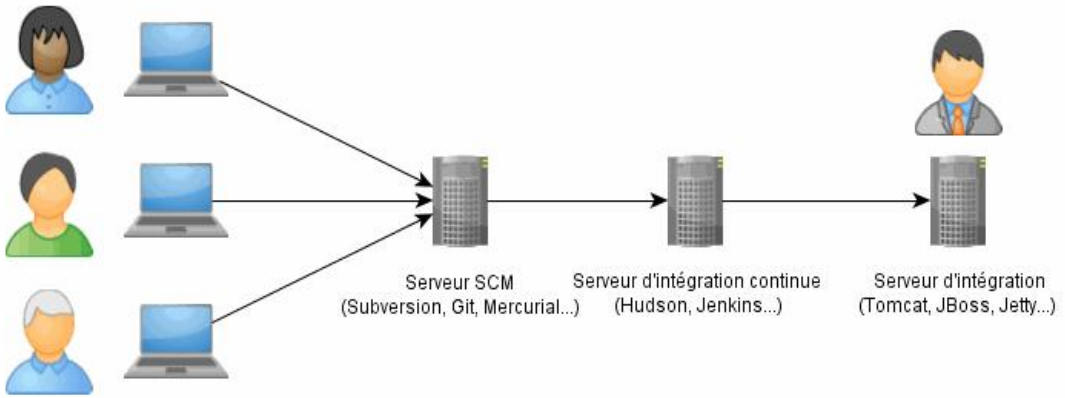
Bonnes pratiques

- Respect du principe d'encapsulation (loi de Déméter)
  - Un composant ne peut modifier que ses enfants directs
  - Pas de modification de son parent direct
  - Pas de modifications de ses petits-enfants directement
- Pour signaler des actions à faire sur son composant parent, il est possible d'envoyer un événement personnalisé
  - Le parent écoute l'événement et effectura les modifications voulues lors de celui-ci
  - Utilisation d'un bus d'événement en cas de hierarchie complexe de composants / ou d'événement à écouteurs multiples
- Un composant gère la navigation par actions sur ses enfants directs, qui répercutent cette navigation sur les leurs, etc

USINE LOGICIELLE

Une intégration souple

- JavaFX dispose d'un plugin Maven
- Principe de l'intégration continue
  - Le livrable est construit sur le serveur d'intégration continue
  - Déploiement automatique possible sur un serveur d'intégration des versions
  - Signalement automatique des erreurs de builds



Maven et JavaFX

Description du plugin

- Support des différents types de déploiement JavaFX
  - Avec ou sans JVM
  - Web ou client lourd
- Un archetype est disponible pour créer un patron d'application JavaFX

  mvn archetype:generate -DarchetypeGroupId=com.zenjava
  -DarchetypeArtifactId=javafx-basic-archetype
  -DarchetypeVersion=1.1
- Possibilité de signer les livrables
- http://zenjava.com/javafx/maven/