

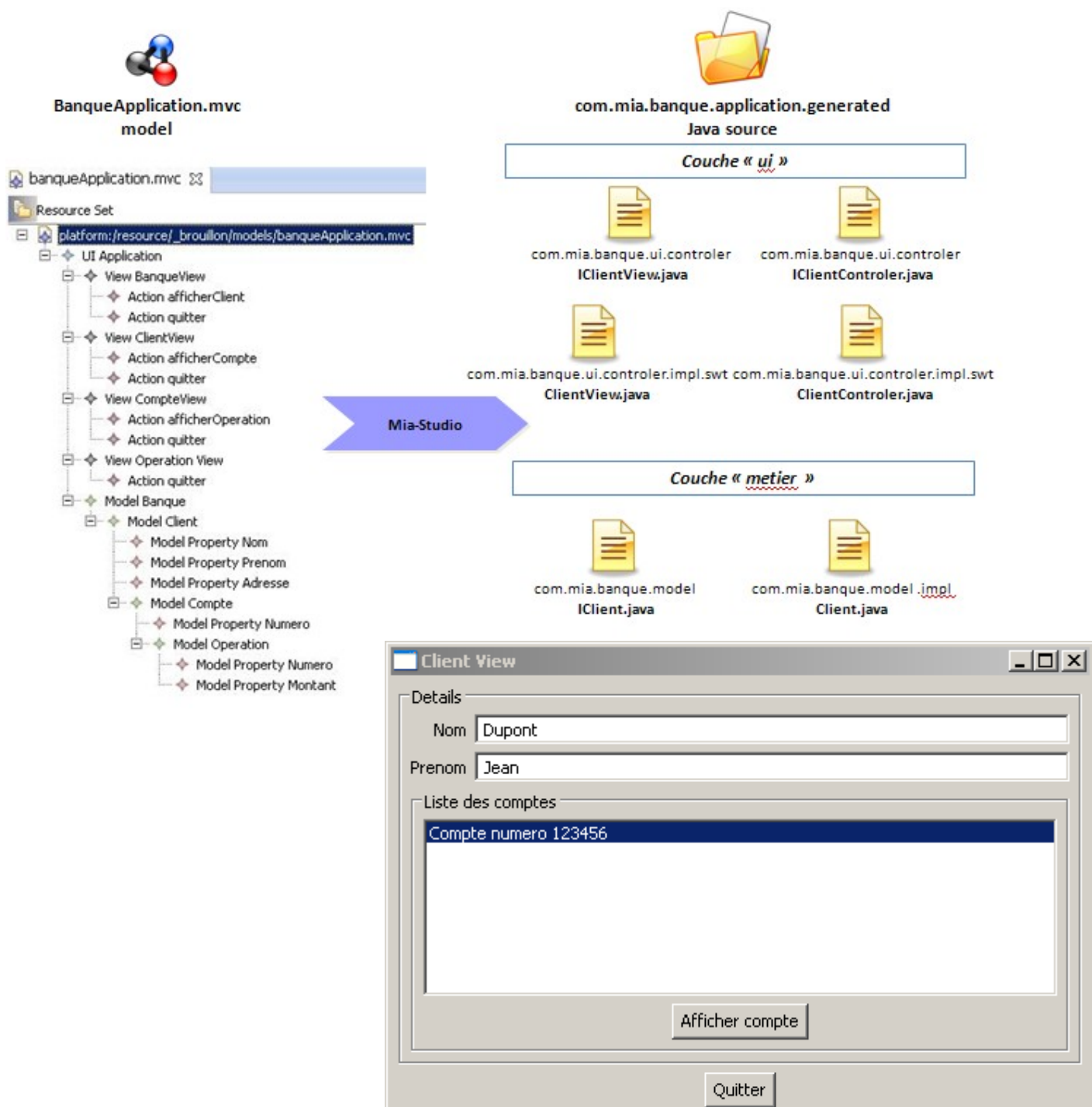
# Formation MDE-Generation – M2 2013-2014 – Projet autonome

Fabien Giquel, Mia-Software - fgiquel@mia-software.com

## Objectif

Ecrire un generateur de code partant d'un modèle de couche présentation de type UI "MVC". Le code applicatif généré aura les caractéristiques suivantes :

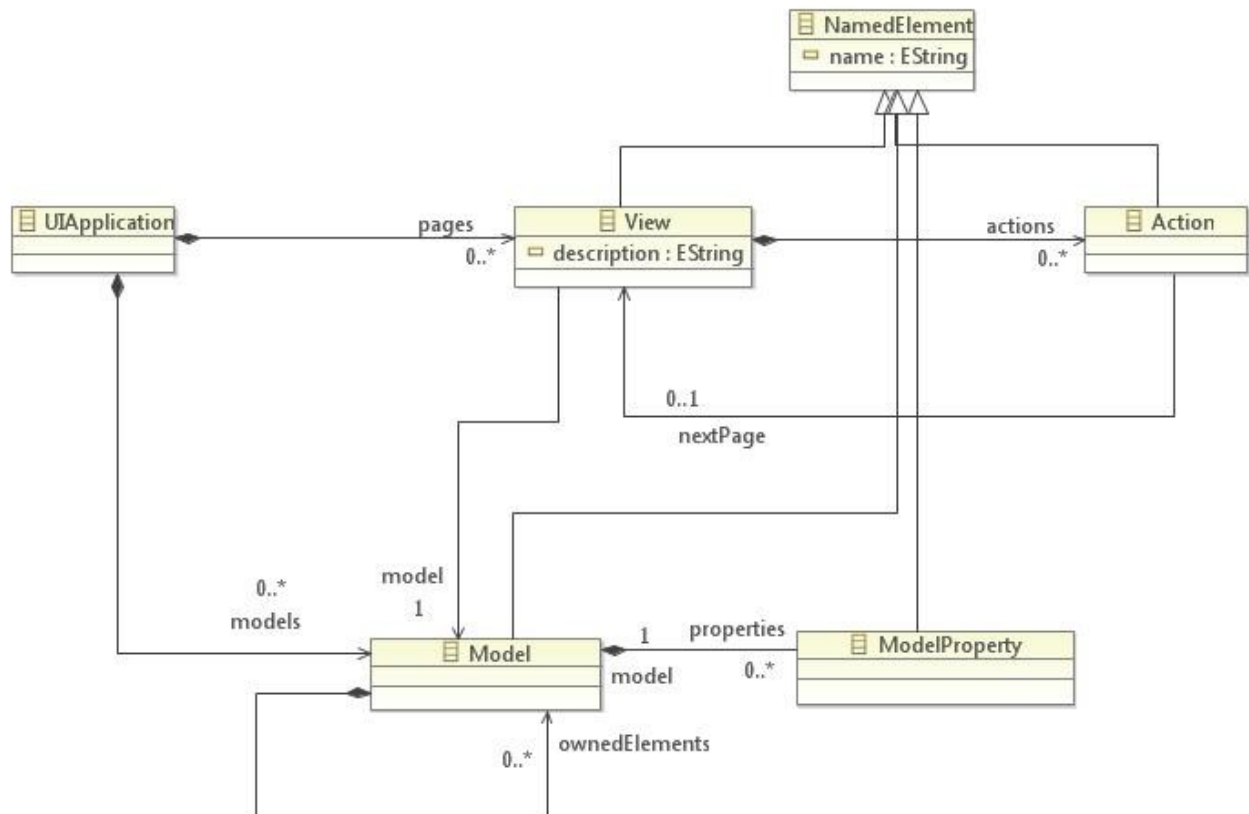
- Couche metier : generation de couple interface/classe pour chaque classe "model"
- Application d'une variante du pattern MVC, avec separation contrat/implementation sur une technologie
  - Generation d'une interface java XXXView pour chacune des vues modélisées
  - Generation d'une interface java XXXControler pour chacune des vues modélisées
  - Generation d'une implementation en SWT de ces interfaces



## Fournitures

1 méta-modèle

- **com.mia.formation.mvc.metamodel**



1 modèle

- **banqueApplication.mvc**

2 applications (projet de type plugin Java) sont fournies

- **com.mia.banque.application.reference**: un exemple de l'application SWT cible illustrant ce que l'on doit générer
- **com.mia.banque.application.tests** : une classe permettant d'initialiser des instances banque/compte/client d'exemple et de démarrer le premier écran de com.mia.banque.application

L'objet final du générateur est de générer le code d'un projet "**com.mia.banque.application.generated**" analogue à "**com.mia.banque.application.reference**".

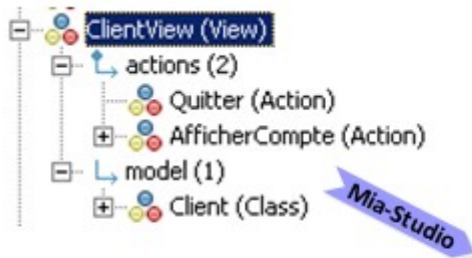
Le projet de type plugin java "**com.mia.banque.application.generated**" devra donc être initialisé vide.

**Attention** : le projet **com.mia.banque.application.reference** est un prototype illustrant la cible de génération. Il ne faut pas générer exactement le même code. Pour les besoins de génération le code des classes peut être réorganisé (exemple : sortir les variables "Button" des méthodes pour en faire des variables d'instance).

/

## Illustration de l'application du pattern MVC dans le code

*Pour simplifier la compréhension, les copies d'écran ne montrent qu'une partie du code source final des classes Java.*



```
IClientView.java IClientController.java
package com.mia.banque.ui.view;

public interface IClientView {

    public void afficherCompte();

    public void quitter();

}
```

```
IClientView.java IClientController.java
package com.mia.banque.ui.controller;

public interface IClientController {

    public void initView();

    public void afficherCompte();

    public void quitter();

}
```

```
IClientView.java IClientController.java ClientView.java ClientController.java
package com.mia.banque.ui.view.impl.swt;

import org.eclipse.swt.SWT;

public class ClientView extends Shell implements IClientView {

    private Client model;
    private IClientController controller;

    private Button btnAfficherCompte;
    private Button btnQuitter;

    public void afficherCompte() {
        this.controller.afficherCompte();
    }

    public void quitter() {
        this.controller.quitter();
    }

    /**
     * Create the shell.
     *
     * @param display
     */
    public ClientView(Display display, Client model, IClientController controller) {

    }

    /**
     * Create contents of the shell.
     */
    protected void createContents() {

    }
}
```

```
IClientView.java IClientController.java ClientView.java ClientController.java
package com.mia.banque.ui.controller.impl.swt;

import org.eclipse.swt.widgets.Display;
import com.mia.banque.ui.controller.IClientController;
import com.mia.banque.ui.view.impl.swt.ClientView;
import data.client.Client;

public class ClientController implements IClientController {

    private ClientView view;
    private Client model;

    public ClientController(Client model) {
        this.model = model;
    }

    @Override
    public void initView() {
        Display display = Display.getDefault();
        this.view = new ClientView(display, this.model, this);
        this.view.open();
        this.view.layout();
        while (!this.view.isDisposed()) {
            if (!display.readAndDispatch()) {
                display.sleep();
            }
        }
    }

    public void afficherCompte() {
        // Start of User Code for afficherCompte
        // End of User Code for afficherCompte
    }

    public void quitter() {
        // Start of User Code for quitter
        // End of User Code for quitter
    }
}
```

## Étapes proposées (pas une obligation)

1. Mise en place de la generation de la "couche metier"
  - Mise en place des scripts de generation des interfaces "IXXX" correspondant aux instances de "Model"
  - Mise en place des scripts de generation des classes "XXX" correspondant aux instances de "Model"
  - Mise en place d'un scenario "model generation" appelant les file templates de generation ci-dessus.
2. Mise en place de la generation de squelettes d' interfaces "IXXXView" (juste la declaration de classe Java, ex : IClientView)
3. Generation dans "IXXXView" des signatures de méthodes pour chaque action liée à la vue
4. Generation de squelettes d' interfaces "IXXXControler" avec les signatures de méthodes pour chaque action liée à la vue et la signature de la méthode "initView()"
5. Generation de squelettes de classes "XXXView" implémentant "IXXXView" et étendant "Shell" (swt). Ces classes sont générées dans un autre package Java suffixé .swt
6. Generation dans "XXXView" d'une variable d'instance stockant le controler correspondant  
ex : `private IClientControler controler;` --> dans ClientView
7. Generation dans "XXXView" d'une variable d'instance stockant le "modèle" correspondant  
ex : `private Client model;`
8. Generation dans "XXXView" d'un constructeur prenant en paramètre une instance de Display (swt), + une instance de XXXControler correspondante + une instance de XXX (classe metier) correspondante
9. Generation dans "XXXView" des méthodes correspondant aux actions, dont le corps se contente de déléguer aux mêmes méthodes définies sur le controler
10. Generation dans "XXXView" de variables d'instances de type "Button" (SWT) pour chacune des actions
11. Generation dans "XXXView" d'une méthode "createContents()" appelée par le constructeur. Cette méthode initialise les boutons dans le shell et leur listener (appel des méthodes d'action)
12. Complement de la méthode "createContents" pour initialiser les widgets présentant les infos du "modèle" (exemple : numéro de compte, nom de client, ...). Laisser de côté les infos de type liste.
- ...
13. Generation des classes XXXControler => à vous de jouer ... Le corps des méthodes d'actions sera laissé vide avec des "balises" Mia-Studio (//Start Of... //End Of) pour une implementation manuelle

## Critères de réussite

La complétude du générateur, étant donné le temps imparti limité, n'est pas un critère essentiel.

Plus importants sont :

- la compréhension générale du passage d'une application d'exemple cible écrite manuellement, à un générateur de code donnant à peu près le même résultat.
- la compréhension et l'application du principe de privilégier les "templates" plutôt que le code "java"
- conception des scripts : la bonne séparation des scripts de generations dans des packages différents par responsabilité technique