# *APES (APES is a Process Engineering Software)*

## *Document d'Architecture Logicielle*

Version 4.0

Auteur : Lionel Petit

# Table des Révisions

| Révision | Date | Auteur(s) | Description |
|---|---|---|---|
| 4.0 | 22/01/2004 | Lionel Petit | Ajout de diagrammes, mise à jour de la couche modèle |
| 2.0 | 29/11/2003 | Lionel Petit | Création du document |

# Table des Matières

# Chapitre 1
# Introduction

## 1. Objectif

Ce document a pour but de dégager et expliquer l'organisation et la conception interne du logiciel.

## 2. Portée

Le document d'architecture logicielle est destiné aux membres de l'équipe et aux superviseurs du projet.

## 3. Références

- Document Vision
- Document des cas d'utilisation
- Glossaire
- Document d'architecture de la version précédente d'Apes

# Chapitre 2
# Structure

## 1. Présentation générale

La structure globale de l'application est une organisation du type MVC (Modèle/Vue/Contrôleur). Le principal but de cette architecture est le découplage entre le modèle et la présentation de ce modèle dans l'application.

De plus, nous essayons de nous focaliser sur une extension aisée des traitements effectués sur le modèle et des interactions possibles entre l'utilisateur et l'application.

## 2. Vue des couches

Cette section présente l'organisation en couches du logiciel. La figure 2.1 montre les dépendances entre ces couches. Elles sont au nombre de quatre et seront détaillées dans les sections suivantes. On explicitera leur rôle et les raisons de leur présence.

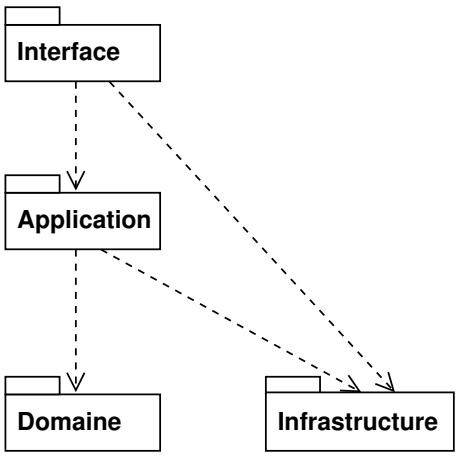


*Figure 2.1: Organisation en couches*

### 2.1. Interface

Cette couche a pour but la présentation des données à l'utilisateur. Elle lui permet aussi d'agir sur le modèle sous–jacent. Grâce à elle, il peut éditer le processus et lancer les opérations disponibles dans la couche Application.

La couche Interface est entièrement couverte par nos classes personnalisées de l'IHM (classes dérivées de l'API Swing).

## 2.2. Application

Cette couche est responsable de la prise en compte des règles de cohérence de l'application et du pilotage de l'application.

La couche Application comporte :
- la validation du modèle
- les classes nécessaires à l'application des interactions utilisateurs sur le modèle
- les actions accessibles depuis la couche Interface (ouverture, sauvegarde...)

C'est cette couche qui est la plus liée aux cas d'utilisation du projet.

## 2.3. Domaine

Cette couche regroupe les classes métiers de l'application et l'implantation des règles de gestion spécifiques.

La couche Domaine est entièrement couverte par notre implémentation du métamodèle SPEM. L'étendue de notre implémentation (par rapport au SPEM complet) est pondérée par les cas d'utilisation de l'application. Plus le client prend en compte d'entités différentes dans ses cas d'utilisation, plus il est nécessaire d'étendre notre modèle.

## 2.4. Infrastructure

Cette couche concerne les composants réutilisés. Elle facilite le développement de trois aspects importants du projet :
- l'affichage et l'interaction avec le diagramme, grâce à JGraph
- l'accès et le stockage des données persistantes, grâce à JSX (qui dérive l'API de sérialisation Java).
- l'importation de composant, grâce à SAX

# 3. Sous systèmes et paquetages

## 3.1. Organisation des paquetages et composants

Pour chaque couche on trouve un ou plusieurs paquetages :

| Couche | Paquetages |
|---|---|
| Interface | apes.ui |
| Application | utils, apes, apes.ui.tools, apes.ui.actions, apes.processing, apes.adapters |
| Domaine | apes.model (et tout ses sous paquetages) |

| Couche | Paquetages |
|---|---|
| Infrastructure | jgraph, JSX, SAX |

Chacun de ces paquetages présente des dépendances, elles sont représentées dans la figure 2.2.

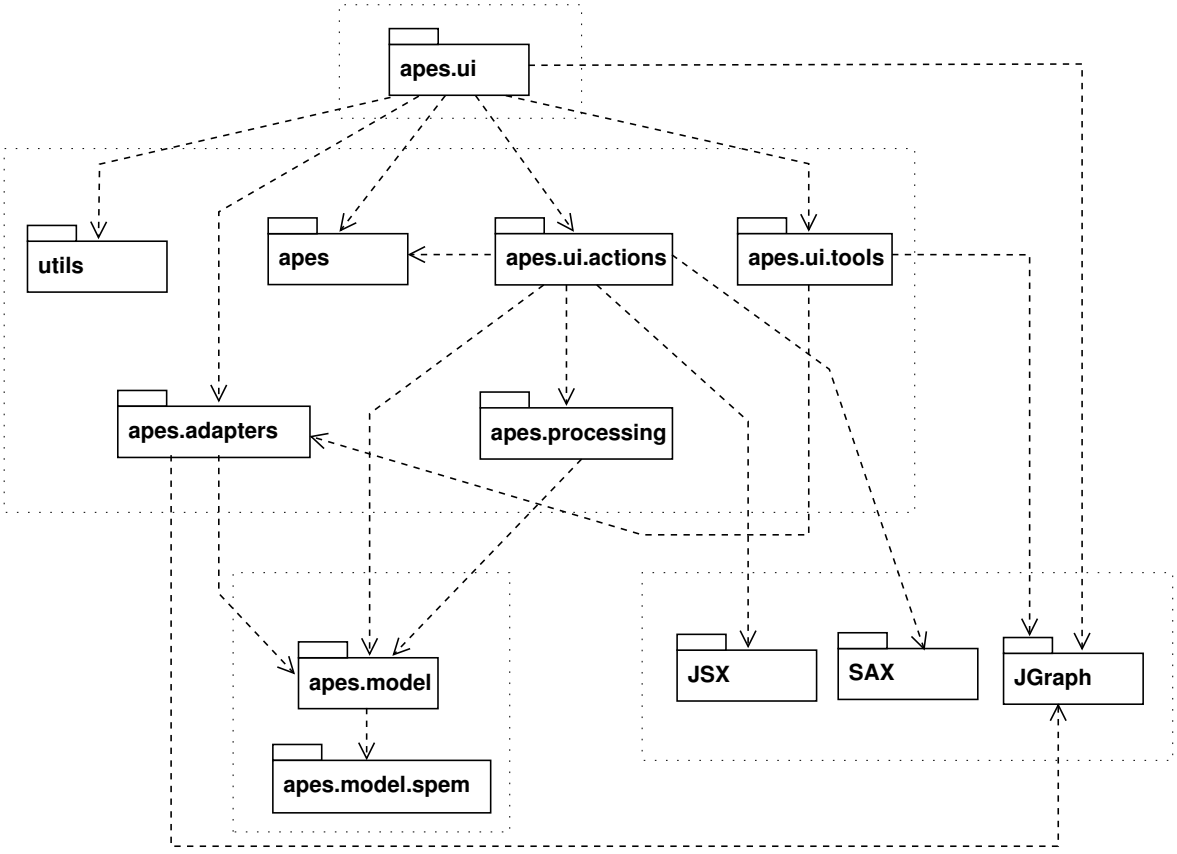


*Figure 2.2: Dépendances des paquetages*

**N.B. :** Parallèlement aux dépendances présentes sur la figure 2.2, tous les paquetages de la couche Application sont dépendants du paquetage utils.

## 3.2. Paquetages développés

### 3.2.1. utils

Ce paquetage fournit des classes utilitaires non spécifiques à l'application. En conséquence, l'ensemble des classes présentes pourront être réutilisées dans d'autres applications.

**Debug**
Classe permettant de conserver des messages de déverminage dans l'application. Elle permet à la compilation, d'activer ou non ses fonctions d'analyse.

**ResourceManager**
Classe facilitant l'internationalisation. Elle permet de charger automatiquement un fichier de messages adapté à la langue de l'utilisateur. Elle dispose d'une instance unique dans l'application, c'est pourquoi le modèle de conception du Singleton est particulièrement indiqué dans son cas.

**IconManager**
Classe facilitant le chargement et la manipulation de ressources graphiques. Elle dispose d'une instance unique dans l'application, c'est pourquoi le modèle de conception du Singleton est particulièrement indiqué dans son cas.

### 3.2.2. apes

Ce paquetage fournit deux classes importantes.

**Context**
Classe représentant à tout instant l'état interne de l'application (principalement l'interface). Elle dispose d'une instance unique dans l'application, c'est pourquoi le modèle de conception du Singleton est particulièrement indiqué dans son cas.

**Project**
Classe représentant un projet complet de l'application. C'est elle qui permet de faire le lien entre le modèle SPEM et la vue de ce modèle au travers des diagrammes.

### 3.2.3. apes.ui

Ce paquetage concerne l'IHM de l'application. Il contient les classes organisant l'aspect graphique de l'application et les interactions de l'utilisateur avec celle–ci.

**ApesFrame**
Fenêtre principale de l'application.

**GraphFrame**
Fenêtre affichant un graphe. Elle dispose de deux méthodes suivants le modèle de conception de la fabrication pour la construction de la palette d'outils associée et de la zone d'affichage du graphe. Elle est donc dérivable suivant les types de graphes affichés.

**ApesTree**
Zone d'affichage du modèle complet sous forme d'arbre (seule la structure des paquetages est affichée ici, les autres relations entre éléments seront visibles dans des GraphFrame).

**ToolPalette**
Palette d'outils de manipulation d'un graphe.

### 3.2.4. apes.ui.tools

Ce paquetage contient tout ce qui est nécessaire à la manipulation des outils utilisés pour l'édition des diagrammes.

**Tool**
Classe de base pour tous les outils de l'application. Elle implémente les opérations nécessaires à la gestion des observateurs ToolListener.

**ToolListener**
Interface observateur adaptée à la classe Tool.

**DefaultTool**
Outil par défaut de l'application. Il permet le déplacement des noeuds du graphe, la suppression d'éléments du graphe...

**CellTool**
Outil dédié à l'ajout de noeuds dans un graphe. Il n'est pas nécessaire de le dériver car il fonctionne par prototypage.

**EdgeTool**
Outil dédié à l'ajout d'arcs dans un graphe. Il n'est pas nécessaire de le dériver car il fonctionne par prototypage.

### 3.2.5. apes.ui.actions

Ce paquetage contient l'implémentation des actions accessibles depuis l'IHM. Cela concerne toutes les opérations du type *"ouvrir un fichier"*, *"quitter"*, *"copier"*... Pour chacune de ces actions, on peut associer un icône et un raccourci clavier.

La totalité des cas d'utilisation nécessitant l'appui d'un simple bouton ou l'utilisation d'un raccourci clavier, ont une classe dans ce paquetage.

### 3.2.6. apes.processing

Ce paquetage contient les classes nécessaires aux traitements sur le modèle SPEM. On trouve deux familles de classes dans ce paquetage, chacune étant inspirée par un modèle de conception :

- les stratégies qui déterminent comment le parcours du modèle sera effectué
- les visiteurs qui fixent les opérations effectuées pour chaque type d'éléments du modèle traité

Par combinaisons d'objets de ces deux familles, on peut simplement obtenir de nouveaux traitements à effectuer sur le modèle.

### 3.2.7. apes.adapters

Ce paquetage contient simplement les classes nécessaires pour adapter le modèle aux besoins de la couche Interface.

**SpemTreeAdapter**
Classe mettant en avant la structure arborescente du modèle (paquetages et éléments de modèle).

**SpemGraphAdapter**
Classe de base nécessaire à l'adaptation du modèle vers un type de diagramme.

### 3.2.8. apes.model

#### 3.2.8.1. apes.model.spem

Ce paquetage contient des sous paquetages conformes au standard SPEM et deux classes suivant le modèle de conception du Visiteur. Les deux classes sont tout d'abord présentées, les sous paquetages de apes.model.spem seront détaillés ensuite.

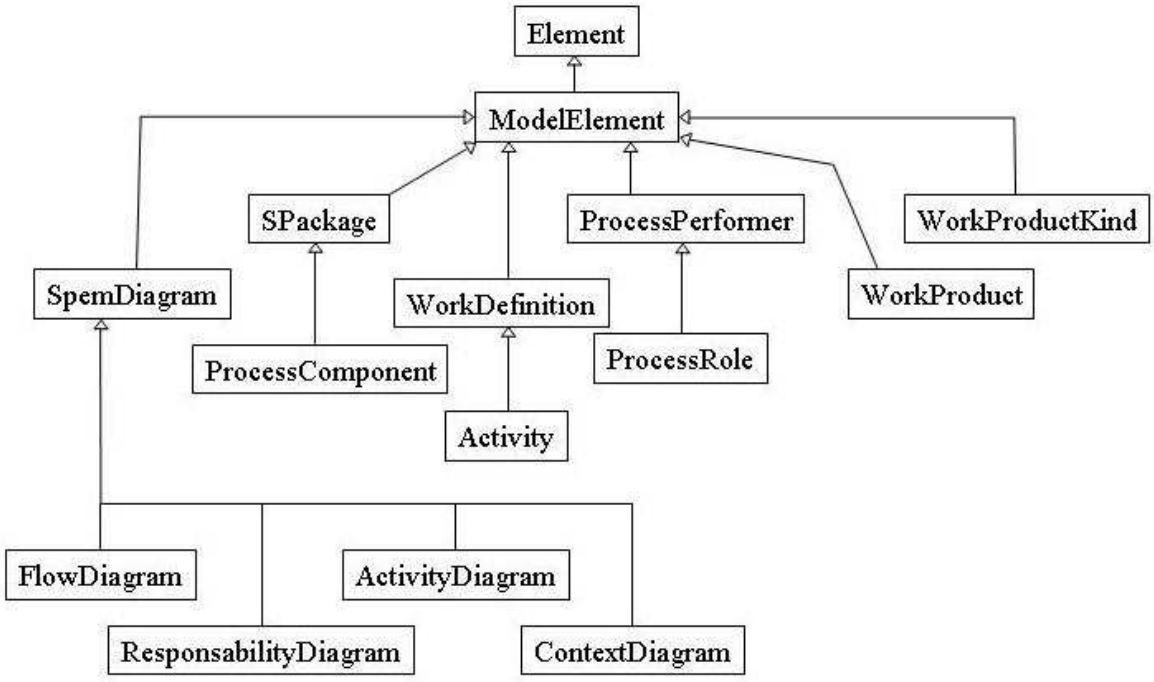


*Figure 2.3: Hiérarchie des classes*

**SpemVisitor**

**DefaultSpemVisitor**

### apes.model.spem.basic

**ExternalDescription**
Classe représentant la description externe d'un élément de modèle.

**Guidance**
Classe dont les instances représentent les guides du modèle.

**GuidanceKind**
Classe permettant de définir des familles de guides.

### apes.model.spem.core

**Element**
Toutes les classes du métamodèle sont filles de cette classe. Elle définit un point d'entrée pour visiteur.

**ModelElement**
Cette classe regroupe tous les objets internes au modèle.

**PresentationElement**
Cette classe regroupe tous les objets documentant des objets du modèle.

### apes.model.spem.modelmanagement

**Package**
Classe permettant de créer des paquetages et permettant donc de regrouper des éléments du modèle.

### apes.model.spem.process.components

**ProcessComponent**
Classe permettant de représenter des composants de processus.

**Process**
Classe dont les instances représentent un processus complet.

### apes.model.spem.process.structure

**Activity**
Classe permettant de représenter les activités du modèle.

**ProcessPerformer**
Classe permettant de représenter les exécutants de processus du modèle.

**ProcessRole**
Classe permettant de représenter les rôles de processus du modèle.

**WorkDefinition**
Classe permettant de représenter les définitions de travail du modèle.

**WorkProduct**
Classe permettant de représenter les produits de travail du modèle.

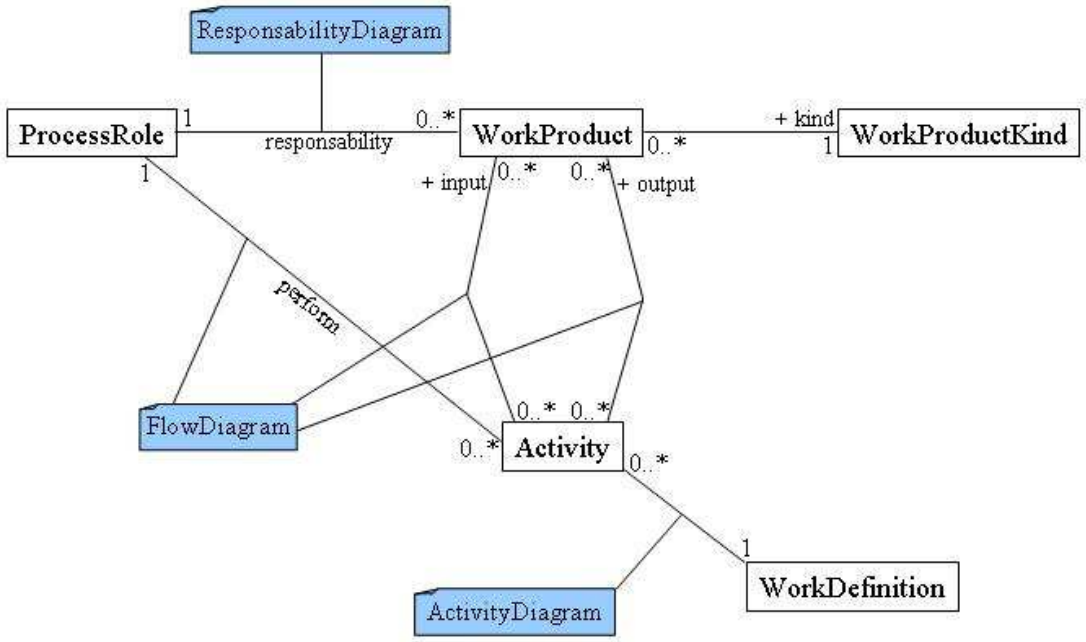


*Figure 2.4: Structure*

### 3.2.8.2. apes.model.extension

Ce paquetage contient l'ensemble des classes qui permettent d'étendre le SPEM. Nous y trouvons les classes permettant de représenter des diagrammes adaptés au SPEM.

**ApesProcess**
La racine du modèle. Elle contient le composant et les interfaces.

**WorkProductRef**
Représente un produit de travail dans une interface

**ApesWorkDefinition**
Surcharge de la définition de travail du spem pour lui ajouter un diagramme de flôt et un diagramme d'activités.

**SpemDiagram**
Classe de base des diagrammes adaptés au SPEM.

**FlowDiagram**
Classe permettant de représenter des diagrammes de flôts permettant de modéliser les relations entre activités, produits de travail et rôles.

**ActivityDiagram**
Classe permettant de représenter des diagrammes d'activités permettant de modéliser le déroulement des activités et du processus.

**ResponsabilityDiagram**
Classe représentant les rôles responsablent des produits de travail.

**ContextDiagram**
Classe permettant de voir les produits de travail requis par le composant et ceux qu'il fournis.

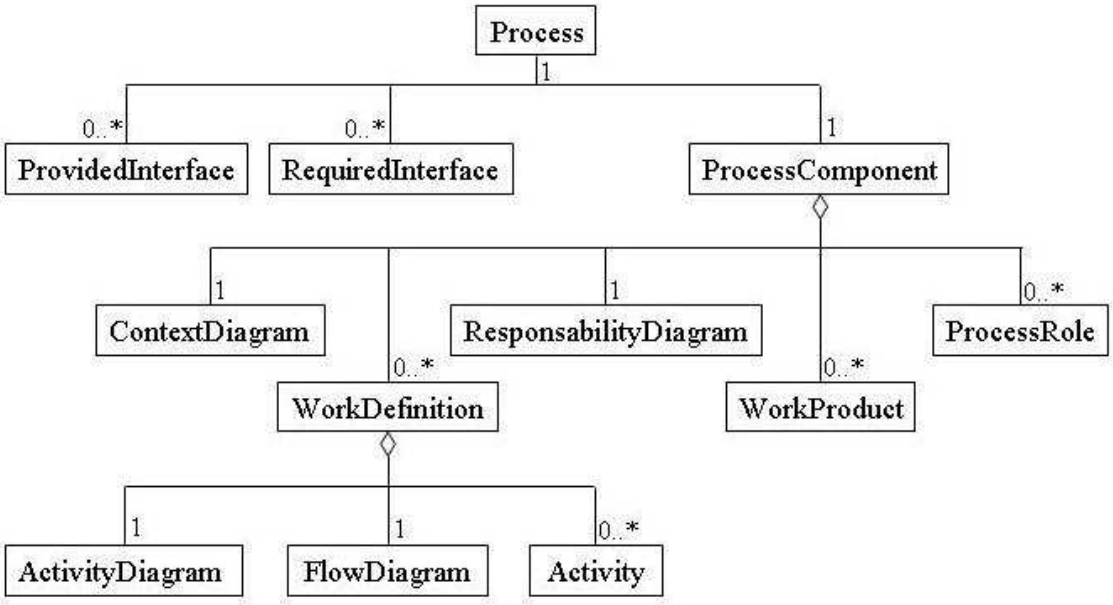


*Figure 2.5: Processus racine*

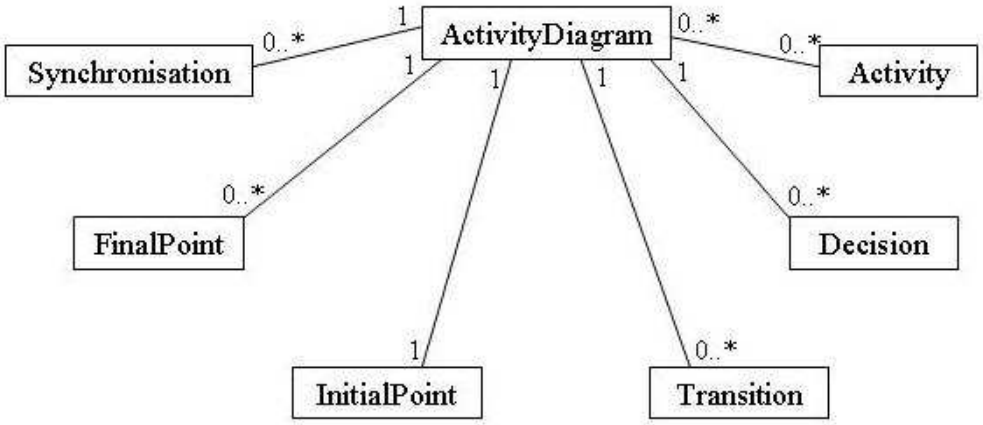


*Figure 2.6: Diagramme d'activité*

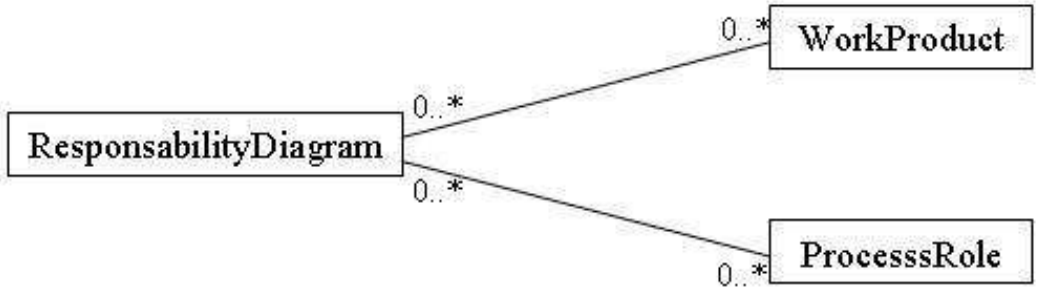


*Figure 2.7: Diagramme de responsabilités*

#### 3.2.8.3. apes.model.frontEnd

Ce paquetage sert à la mise en place d'un médiateur qui centralise les appels de la couche contrôleur vers la couche modèle.

**ApesMediator**
Médiateur qui réceptionne les appels provenant des adapteurs, modifie la couche modèle et envoie une réponse aux objects l'écoutant.

## 3.3. Composants réutilisés

### 3.3.1. JGraph

C'est le composant en charge de l'affichage et de l'édition des graphes du modèle. Il est réutilisé tel quel. Aucune adaptation n'est nécessaire, il suffit d'implémenter nos propres classes personnalisées conformes aux interfaces spécifiées dans le composant.

**JGraph**
Zone d'affichage d'un graphe. C'est la classe centrale du composant (d'où son nom).

**GraphModel**
Interface définissant un modèle compatible pour un JGraph.

**GraphView**
Interface définissant une vue compatible pour un JGraph. Elle permet l'association entre les éléments du modèle et leurs représentants affichés dans le JGraph.

**BasicMarqueeHandler**
Implémentation simple pour gérer les interactions de l'utilisateur avec un JGraph. Il s'agit d'un accès privilégié aux interactions avec la souris.

### 3.3.2. JSX

C'est la librairie en charge de la sérialisation en XML. Elle est basée sur la sérialisation présente dans l'API Java. Elle est donc directement utilisable pour enregister n'importe quel objet Java sérialisable sans modification préalable.

**ObjOut**
Classe permettant la sérialisation XML. Elle pourrait être nommée XMLObjectOutputStream.

**ObjIn**
Classe permettant la désérialisation XML. Elle pourrait être nommée XMLObjectInputStream.

### 3.3.3. SAX

Cette librairie permet d'expoiter le contenu d'un fichier xml.

# Chapitre 3
# Mécanismes

## 1. Visiteur généalogiste

### 1.1. Motivation

Supposons que l'on dispose à l'écran d'une liste d'éléments apparentés. Lorsque l'utilisateur clique sur un des éléments de la liste, on doit afficher un menu contextuel.

Si deux éléments diffèrent uniquement par leur profondeur dans l'arbre d'héritage, les menus résultant doivent avoir un nombre d'entrées communes. En effet, les opérations réalisables sur un objet d'une classe sont aussi réalisables sur les objets d'une de ses filles.

On peut donc utiliser un visiteur pour remplir cette tâche puisqu'il permet une identification de type à l'exécution. Mais en tenant compte en plus des particularités de l'arbre d'héritage, on peut factoriser une partie du comportement du visiteur.

### 1.2. Indications d'utilisation

On utilise le modèle du visiteur généalogiste dans les cas suivants :
- un visiteur classique est applicable, et
- l'ensemble des classes connues de ce visiteur forment un arbre d'héritage complet aux classes abstraites prêt

### 1.3. Implémentation

On part d'une implémentation classique du visiteur. Il suffit simplement d'ajouter un visiteur concret implémenté comme suit :
- on ajoute pour chaque classe abstraite de l'arbre d'héritage complet une méthode de visite protégée
- toutes les méthodes du visiteur ne font qu'une seule chose, appeler la méthode correspondant à la classe parent, sauf la méthode correspondant à la racine de l'arbre d'héritage qui ne fait rien.

Ainsi il suffit de dériver la classe obtenue et surcharger les méthodes ad hoc pour obtenir le comportement attendu, c'est à dire factoriser des comportements en fonction de la branche d'héritage.

### 1.4. Exemple de code

Voici un exemple en Java de ce modèle.

Supposons que nous disposons des classes suivantes :

```
abstract class Element
{
        public abstract void accept(Visitor v);
}

class ElementA extends Element
{
        public void accept(Visitor v) { v.visitElementA(this); }
}

class ElementB extends Element
{
        public void accept(Visitor v) { v.visitElementB(this); }
}

class ElementC extends ElementB
{
        public void accept(Visitor v) { v.visitElementC(this); }
}
```

Alors l'implémentation d'un visiteur généalogiste donnera ceci :

```
interface Visitor
{
        public void visitElementA(ElementA a);
        public void visitElementB(ElementB b);
        public void visitElementC(ElementC c);
}

class GenealogistVisitor implements Visitor
{
        public void visitElement(Element e) { }
        public void visitElementA(ElementA a) { visitElement(a); }
        public void visitElementB(ElementB b) { visitElement(b); }
        public void visitElementC(ElementC c) { visitElementB(c); }
}
```

## 1.5. Utilisations remarquables dans APES

La classe `DefaultSpemVisitor` du paquetage `apes.spem` est conforme à ce modèle. Ses filles sont notamment utilisées dans l'arbre pour afficher les icônes associés à ses noeuds et pour associer un menu contextuel à chaque noeud.

# Chapitre 4
# Qualité de l'architecture

## 1. Avantages

Le principal avantage de cette architecture est une réutilisation forte de composants éprouvés dans leur domaine. De plus, ces composants sont activement maintenus, et sont utilisés dans d'autres applications.

Il est important de noter la constitution d'un patrimoine des classes réutilisables par le biais du paquetage utils.

Ensuite, de nouveaux traitements sur le modèle peuvent être écrits facilement grâce à l'organisation du paquetage apes.processing et à l'utilisation du modèle de conception du visiteur dans notre implémentation du SPEM.

Enfin, un des principaux atouts de notre architecture est le couplage faible entre notre implémentation du SPEM et le reste de l'application.

## 2. Inconvénients

Cette architecture présente quelques inconvénients. Le plus flagrant provient d'une limitation du composant JGraph. En effet, ce dernier impose une relation 1/1 entre les éléments du GraphModel et les éléments affichés dans le JGraph. Notre modèle SPEM ne peut donc pas implémenter directement l'interface GraphModel. Toutefois, cet inconvénient a une importance limitée puisque l'implémentation directe de l'interface GraphModel augmenterait le couplage entre notre modèle et JGraph.

Ensuite, il nous faut écrire des adaptateurs entre le modèle SPEM et JGraph d'une part, et entre le modèle SPEM et JTree d'autre part. Ces adaptateurs sont un travail important de l'architecture et sont lourds à écrire si le modèle sous jacent est éloigné de l'interface à obtenir.

Enfin, l'utilisation du modèle de conception du visiteur dans le paquetage apes.processing peut provoquer l'apparition de classes fastidieuses à écrire si le modèle SPEM implémenté devient trop étendu.

## 3. Extensions possibles

Le modèle SPEM implémenté dans APES peut être étendu et modifié. Il peut être important de le rendre observable (en plus de visitable), afin de réduire encore plus son couplage avec le reste de l'application. Il faudra, toutefois, évaluer l'impact d'un tel remaniement sur les performances globales de l'application (le fait que les adaptateurs du modèle soient eux même observables

devrait normalement suffire).

L'utilisation du prototypage dans les classes EdgeTool et CellTool ainsi que la présence de méthodes fabrications dans GraphFrame permettent de facilement augmenter le nombre de types de diagrammes différents.

# Chapitre 5
# Principales évolutions

Ce chapitre présente les évolutions majeures effectuées sur l'architecture d'Apes.

## 1. JGraph

La version utilisée de JGraph est passée de la 1.* à la 3. Ceci a engendré de nombreux changements au niveau de l'utilisation des graphes.

## 2. Communication entre les couches contrôleur et modèle

Pour simplifier les communications entre l'arbre, les graphes et le modèle, un médiateur a été mis en place. Pour effectuer une action sur le modèle, l'adapteur doit envoyer une commande contenant sa requête au médiateur. Celui–ci vérifie que l'action est possible et, le cas échéant, envoie un message aux différents objets qui sont à son écoute.

Il en résulte que les adapteurs ne peuvent modifier le modèle directement.

Les avantages :
- découplage entre l'arbre et les graphes
- centralisation des actions
- clarification de la communication

Inconvénients :
- une mauvaise implémentation pourrait rendre la maintenance du médiateur difficile

Diagramme de sèquence de l'insertion d'un élément dans l'arbre :

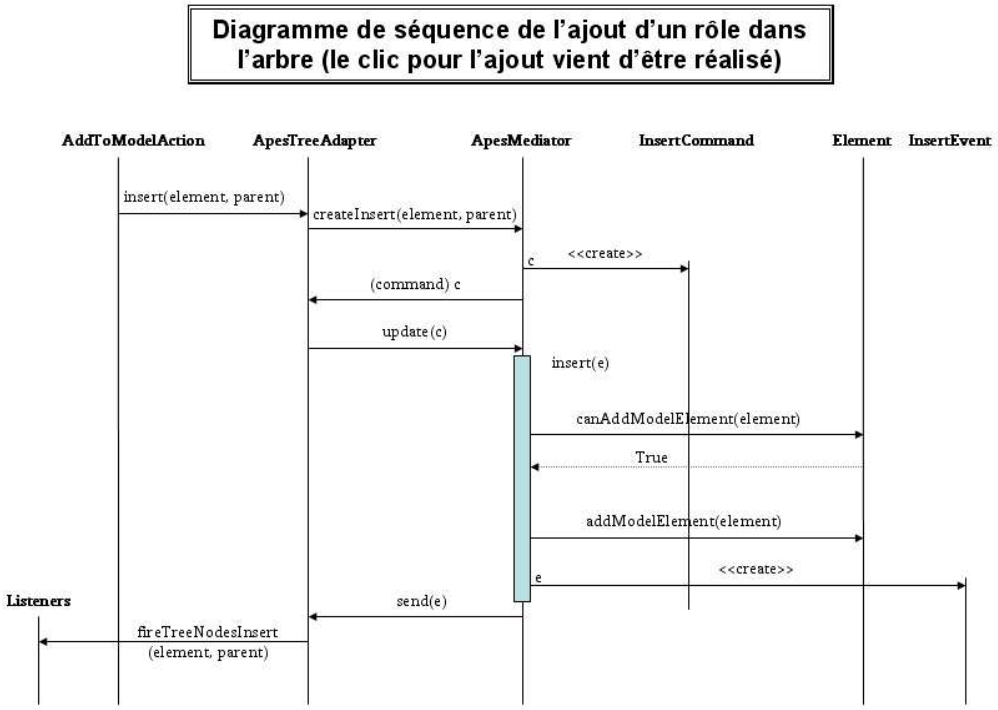


*Figure 5.1: Diagramme de séquence*