Scénarios, tactiques et vues architecturales

Modificabilité

[**1 Le GAB peut être utilisé par différent systèmes bancaires (différentes entreprises).**](#_fboqi229x7ja) **3**

[1.1 Tactique 1 : Utiliser un intermédiaire](#_lu1e5w1kapvx) 3

[1.2 Vue architecturale (à l’extérieur de la classe)](#_jpk7s8xak57m) 3

[1.2.1 Description du diagramme](#_30j0zll) 4

[1.2.2 Description des éléments](#_1fob9te) 4

[1.2.3 Relation entre éléments et tactiques](#_3znysh7) 4

[**2 Scénario : Changement de l’interface utilisateur (Shell) du guichet selon les préférences de la banque**](#_ok2qvvcio7ly) **5**

[2.1 Tactique 1 : Utilisation d’un intermédiaire](#_b2q1k16vw5vp) 5

[2.2 Tactique 2 : Restriction des dépendances](#_r6sydxwa5psl) 5

[2.3 Vue architecturale (à l’extérieur de la classe)](#_ep28vi1u1f7c) 6

[**3 Le guichet peut supporter les clients des autres banques et leur permettre de faire des retraits.**](#_6kl2y5v4bga9) **7**

[3.1 Tactique 1 : utiliser un intermédiaire(patron adaptateur)](#_8bb1v5ynrdjj) 7

[3.2 Tactique 2 : refactoring (classes abstraites)](#_yqj22qoz1co4) 7

[3.3 Tactique 3 :Abstract common services](#_dz1736xp0vo6) 7

[3.4 Diagramme:](#_es97uubadyle) 8

[3.4.1 Description du diagramme](#_ymexhfsf1zwr) 8

[3.4.2 Table de description des éléments du diagramme](#_hfmgsbcntgl9) 8

[3.4.3 Description de la relation entre les éléments et les tactiques](#_m05ul5h6ss9u) 9

[**4 Scénario : La banque est en pleine expansion et nous avons des guichets qui ouvrent un peu partout, la banque veut donc réduire les coûts d’installation/configuration du système.**](#_xxh6e295hb90) **10**

[4.1 Tactique 1 : Encapsulation](#_4ykbvn8bq4cz) 10

[4.2 Tactique 2 : Abstract Common Services](#_a8pw7jitcqp) 10

[4.3 Tactique 3 : Defer Binding](#_dg065h1pvypg) 10

[4.4 Vue architecturale (à l’extérieur de la classe)](#_3w8w21q63u6w) 11

[4.4.1 Diagramme](#_2et92p0) 11

[4.4.2 Diagramme de séquence](#_tyjcwt) 12

[4.4.3 Légende](#_qaouqeh6du1u) 12

[4.4.4 Texte de description du diagramme](#_5s2y6iuddkpq) 12

[4.4.5 Table de description des éléments du diagramme](#_4zmmv6vdvtsm) 13

[4.4.6 Texte décrivant la relation entre les éléments et les tactiques](#_crydytnjjjcv) 13

[**5 Scénario :**](#_8cemfa2008h0) **14**

[5.1 Tactique 1 : Module divisé](#_6jhe8oorjlwe) 14

[5.2 Tactique 2 : Encapsulation](#_eep7g125xebx) 14

[5.3 Tactique 3 : Augmenter la cohérence sémantique](#_i2ydorhtxdvg) 14

[5.4 Diagramme 1.0 Modélisation de la modificabilité du système d'ATM.](#_6jm9h79vcvpr) 15

[5.4.1 Description du diagramme:](#_2ha8zzlk46gp) 15

[5.4.2 Table de description des éléments du diagramme :](#_56n1zpb0rty4) 15

[5.4.3 Relations :](#_9ng6q9wh0108) 15

[**6 Scénario : Modifiabilité**](#_w7cree196o02) **16**

[6.1 Tactique 1 :](#_n0yw4x1e51og) 16

[6.2 Vue architecturale (à l’extérieur de la classe)](#_fmrdep2w7ksc) 17

[**7 Scénario : Modifiabilité**](#_x0jozeu9mr4c) **18**

[7.1 Tactique 1 : Reduce size of a module - Split module](#_2v3wm818we1s) 18

[7.2 Tactique 2 : Reduce coupling - Encapsulate](#_qxjxfab7hgu) 18

[7.3 Tactique 3 : Reduce coupling - Use an intermediary](#_n5sqtw149udd) 18

[7.4 Vue architecturale (à l’extérieur de la classe)](#_4zdwfi586cbh) 19

[7.4.1 Diagramme](#_2s8eyo1) 19

[**8 Scénario :**](#_2z6i3ae2bs25) **20**

[8.1 Tactique 1 : Encapsulation](#_enxyd6rd9pvf) 20

[8.2 Tactique 2 : Restriction des dépendances](#_z2k7m51siaka) 20

[8.3 Tactique 3 : Adaptateur](#_6kp82flnrvuo) 20

[8.4 Vue architecturale (à l’extérieur de la classe)](#_v8l92ziz0aqb) 20

[8.4.1 Diagramme](#_17dp8vu) 20

[8.4.2 Légende](#_rvayoq19c7lm) 21

[8.4.3 Texte de description du diagramme](#_1ifdsnk7rqqd) 21

[8.4.4 Table de description des éléments du diagramme](#_7bxfay5iex7s) 22

[8.4.5 Texte décrivant la relation entre les éléments et les tactiques](#_k3iuk59ynhdz) 22

[**9 Scénario :**](#_inztk7516gnj) **23**

[9.1 Tactique 1 : Encapsulation](#_2n2uese33zzx) 23

[9.2 Tactique 2 : Abstract Common Services](#_o3mtv5d7j0hq) 23

[9.3 Vue architecturale (à l’extérieur de la classe)](#_8993enw3ve8f) 24

[9.3.1 Diagramme](#_3rdcrjn) 24

[9.3.2 Légende](#_i43mr78qbell) 24

[9.3.3 La classe abstraite AbstractMoney implémente l’interface MoneyInterface](#_gbwdm0yo3ouw) 24

[9.3.4 Texte de description du diagramme](#_htz4muq9mas2) 24

[9.3.5 Il faudra créer une classe abstraite nommée AbstractMoney qui implémentera les fonctionnalités communes à la majorité des monnaies. Chaque monnaie étendra cette classe et pourra définir ses particularités (nombre de décimales, symbole, coupures, etc). La classe AbstractMoney implémentera aussi l’interface MoneyInterface, qui définira les services à offrir au système de l’ATM pour chaque monnaie. Le reste du système sera modifié pour utiliser cette interface.](#_78q8xaieslhh) 24

[9.3.6 Table de description des éléments du diagramme](#_6k5qkjlyn52j) 24

[9.3.7 Texte décrivant la relation entre les éléments et les tactiques](#_kvrm144sy06v) 25

# 1 Le GAB peut être utilisé par différent systèmes bancaires (différentes entreprises).

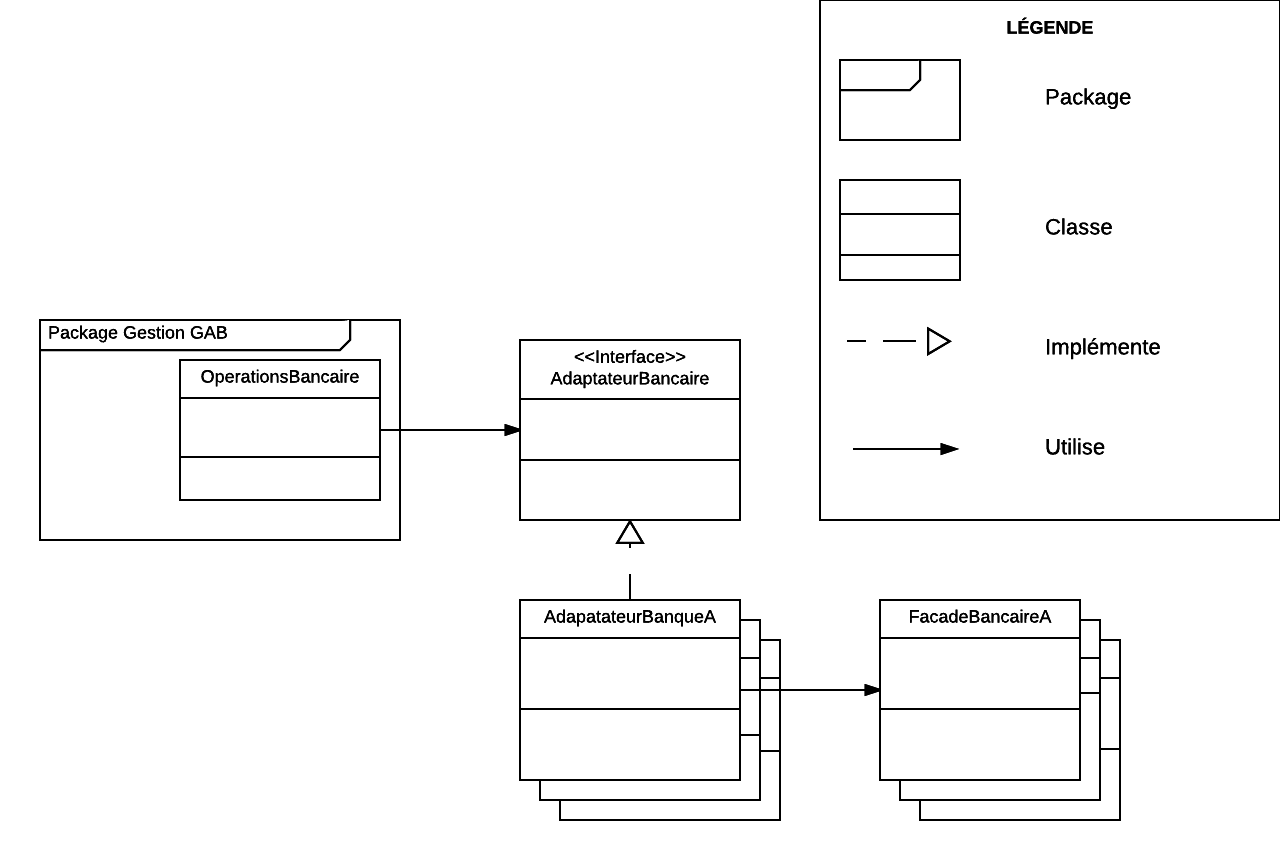
| **objectifs d'affaires** | On doit pouvoir changer le système bancaire lié au GAB en moins de 3 heures. |
| --- | --- |
| **Source** | Développeur |
| **Stimulus** | Nouveau client -- Directive d’une nouvelle banque à traîter |
| **Artéfact** | Code |
| **Environnement** | Conception |
| **Réponse** | Faire une modification |
| **Mesure de la réponse** | Temps et coût à effectuer le changement de système bancaire |
| **Questions** | 1. Quel sont les différentes interfaces des différents systèmes bancaires? |

## 1.1 Tactique 1 : Utiliser un intermédiaire

**Description**: L’utilisation d’une interface suivant le patron adaptateur sera implémentée pour lier le système du GAB aux différents systèmes bancaires.

**Justification**: L'utilisation du patron adaptateur permettra la modification rapide du système logiciel du GAB. Celui-ci n’aura qu’à implémenter une nouvelle instance de l’adaptateur lors de l’ajout d’un nouveau système bancaire.

## 1.2 Vue architecturale (à l’extérieur de la classe)



### 1.2.1 Description du diagramme

Cette vue module montre un patron adaptateur utilisé pour adapter les système bancaire au système du GAB.

### 1.2.2 Description des éléments

**Package Gestion GAB**: Package contenant le système du GAB.

**OperationsBancaire**: Classe des opérations bancaire.

**AdaptateurBancaire**: Interface du patron adaptateur.

**AdapatateurBanqueX**: Classe du patron adaptateur qui implémente l'interface.

**FacadeBancaireX**: Façade bancaire du système de la banque voulant utiliser le GAB.

### 1.2.3 Relation entre éléments et tactiques

**Tactique utiliser un intermédiaire**:

L'interface "AdaptateurBancaire" ainsi que les "AdaptateurBanqueX" qui en découlent dans le système du GAB représente l'utilisation d'un intermédiaire (adaptateur).

# **2 Scénario : Changement de l’interface utilisateur (Shell) du guichet selon les préférences de la banque**

| **Objectifs d'affaires** | Répondre plus efficacement et plus rapidement aux exigences des différentes banques |
| --- | --- |
| **Source** | Développeurs |
| **Stimulus** | * Demande de changement de la part d’une banque * Installation d’un nouveau guichet |
| **Artéfact** | * Code * Interface |
| **Environnement** | * Build time * compile time * design time |
| **Réponse** | * Effectuer la modification * Tester la modification * Déployer la modification |
| **Mesure de la réponse** | * Introduction de nouvelles défectuosités * Temps de mise en place du changement * Coût de développement d’une nouvelle interface utilisateur |
| **Questions** | 1. Est-ce que certaines banques ont des fonctions exclusives ? 2. Quel budget une banque possède-t-elle pour une telle modification ? 3. Quel serait le délai exigé pour cette modification ? 4. Les banques ont-elles une équipe de design interne ? |

## 2.1 Tactique 1 : Utilisation d’un intermédiaire

**Description** : Utiliser une interface entre le *core* de l’application et le *UI*.

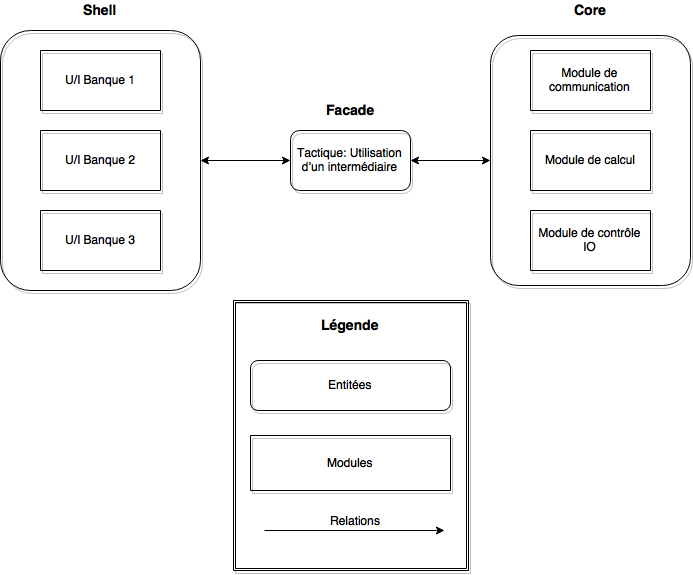
**Justification** : Ainsi, toute nouvelle implémentation de design n’aura qu’à utiliser cet intermédiaire et n’affectera en aucun cas les autres fonctionnalités de l’application.

## 2.2 Tactique 2 : Restriction des dépendances

**Description** : Les dépendances entre chaque vue et le *core* de l’application compliquent énormément la modification du *UI*.

**Justification** : Les modifications au niveau des fonctions de base du *UI* (non spécifique aux banques) ne vont pas affecter les différentes vues spécifiques aux banques.

## 2.3 Vue architecturale (à l’extérieur de la classe)



**Composantes:**

Interface utilisateur spécifique à une banque :

*Shell* qui se change selon les préférences de la banque. Il y a des différents “U/I banque” qui utilisent la façade intermédiaire afin de déterminer laquelle est utilisée dans le Shell.

Core:

Core qui gère le module de communication, module de calcul et module de contrôle “IO”. Il est séparé indépendamment du Shell avec une façade. Cependant, il ne communique pas directement avec le Shell pour changer l’interface usager.

Façade (intermédiaire):

Façade agit comme une intermédiaire entre le Shell et le Core.

# 3 Le guichet peut supporter les clients des autres banques et leur permettre de faire des retraits.

| **objectifs d'affaires** | La banque veut que les clients de diverses banques puissent effectuer des opérations bancaires sur leurs guichets. |
| --- | --- |
| **Source** | Le développeur |
| **Stimulus** | ~~Le développeur~~ a reçu la directive de modifier le fonctionnement du guichet afin de pouvoir supporter les autres banques. |
| **Artéfact** | Le code , module de communication avec la banque |
| **Environnement** | Design time |
| **Réponse** | Implémenter , tester et déployer la fonctionnalité |
| **Mesure de la réponse** | * Temps de modification : 2 semaines * Complexité : complexe * Affecte le comportement de guichet si le client fait partie d’une autre banque |
| **Questions** | 1. Quel type d’opérations le client peut-il faire s’il n’est pas client de la banque? 2. Combien de banques différentes est-ce que le système peut supporter? 3. Combien d’argent est-ce que le client de l’autre banque peut retirer? |

## 3.1 Tactique 1 : utiliser un intermédiaire(patron adaptateur)

**Description**: Implémentation d’une interface permettant de relier les autres interfaces des banques et utilisation d’un patron adaptateur.

**Justification**: L’implémentation d’une interface permettra d’améliorer la modifiabilité, car une interface change peu et doit être respectée. De plus, le patron adaptateur créera le pont entre l’interface de l’ATM et celle des autres banques, ce qui permettra aussi de pouvoir ajouter de nouvelles banques plus facilement à l’avenir.

## 3.2 Tactique 2 : refactoring (classes abstraites)

**Description**: Créer des classes abstraites avec actions de base pour les banques et types de transactions.

**Justification**: Permettra d’ajouter des banques plus facilement et permettra la possibilité d’ajuster une action pour une banque en particulier si celle-ci diffère du comportement de base, tout en limitant le couplage.

## 3.3 Tactique 3 :Abstract common services

**Description**: Créer des classes abstraites avec actions de base pour les banques et types de transactions.

**Justification**: On crée cette classe afin de pouvoir répondre aux comportements différents que peuvent avoir les différentes banques face aux transactions bancaires. Avec la classe abstraite, les différents comportements face aux opérations bancaires( dépôt,retrait transfert) seront couverts dans des sous-classes.

## 3.4 Diagramme:



**Légende**

Diagramme UML. Pas besoin de légende.

### 3.4.1 Description du diagramme

Le diagramme est une représentation exhaustive de la tactique #1 pour l’utilisation d’une interface commune pour communiquer avec plusieurs banques afin de pouvoir faire une transaction comme un dépôt ou un retrait. Le diagramme représente aussi la tactique #2 et #3 qui est un refactoring de la classe banque en classe abstraite, afin de pouvoir implémenter des méthodes adaptées pour chacune des banques ajoutées.

### 3.4.2 Table de description des éléments du diagramme

| **Élément** | **Description** |
| --- | --- |
| Banque | Il s’agit d’une classe abstraite qui représente le concept abstrait des différentes banques pouvant communiquer avec le système. Elle implémente le comportement commun à toutes les banques. Les comportements distincts seront traités dans des sous-classes. |
| Interface | L’interface permet de communiquer avec les systèmes des autres banques. Nos classes communiqueront avec cette interface afin d’obtenir les informations qu’elles ont besoin afin de traiter les différentes opérations bancaires comme le retrait le dépôt et le transfert entre 2 comptes. |

### 3.4.3 Description de la relation entre les éléments et les tactiques

Dans notre cas, nous avons implémenté une interface avec laquelle nos classes pourront communiquer afin d’obtenir l’information dont elles ont besoin afin de pouvoir effectuer les opérations bancaires supportées par notre ATM( dépôt,retrait et transfert). Ainsi, l’interface agit comme un intermédiaire entre notre système et le système des autres banques. Elle permet au système de s’adapter aux différents comportements des systèmes des autres banques. De plus, grâce à la classe abstraite banque, nous pouvons implémenter ses divers comportements dans notre système à l’aide de différentes sous-classes. Cela rend ainsi notre système plus flexible et stable. Il serait aussi facile de permettre à une nouvelle banque de joindre notre système.

# 4 Scénario : La banque est en pleine expansion et nous avons des guichets qui ouvrent un peu partout, la banque veut donc réduire les coûts d’installation/configuration du système.

| **objectifs d'affaires** | Réduire le coût des modifications du logiciel de Guichet Automatique. |
| --- | --- |
| **Source** | System Admin |
| **Stimulus** | L’architecture réseau |
| **Artéfact** | configuration du guichet |
| **Environnement** | Runtime |
| **Réponse** | -Test modification -Deploy it |
| **Mesure de la réponse** | L’installation et la configuration du système doit se faire en moin de 2 heures. |
| **Questions** | 1. ~~Quels sont les ports à l’arrière du guichet ?~~ 2. Quel est le protocole de sécurité ? 3. Quel est le matériel et la configuration à installer ? 4. Quel est le temps nécessaire pour ajouter un guichet actuellement? |

## 4.1 Tactique 1 : Encapsulation

**Description**: Tout ce qui touche à la sécurité doit être encapsulé.

**Justification**: La sécurité est critique dans un guichet bancaire, il doit donc être fiable.

## 4.2 Tactique 2 : Abstract Common Services

**Description**: Permet d’utiliser un module abstrait représentant tous les guichets de l'architecture réseau.

**Justification**: Étant donné que tous les guichets ont la même configuration de base, l’utilisation d’un patron abstract common service est optimal puisqu’il permet de créer des façades seulement lorsque nécessaire et ce, de façon peu coûteuse.

## ~~4.3 Tactique 3 : Defer Binding~~

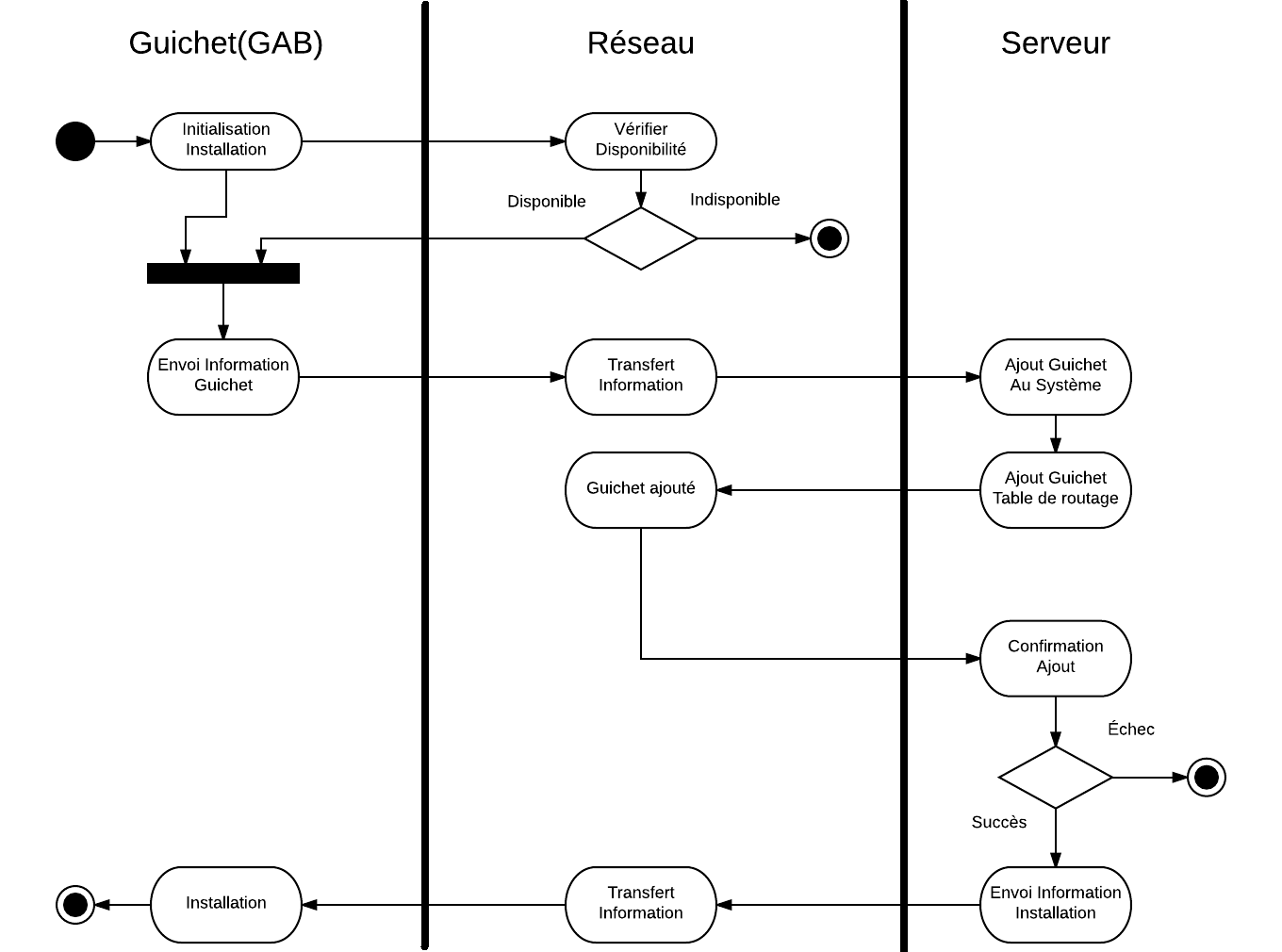
**~~Description~~**~~: Afin de mettre à jour le logiciel de guichet automatique, on ajoute un Defer Binding afin de donner le plus de tâches possible au logiciel derrière afin qu’il puisse se mettre à jour par lui-même sur le réseau.~~

**~~Justification~~**~~: Cette tâche de configuration peut-être coûteuse en temps et en argent, cette automatisation permettra d’effectuer une expansion de la banque plus rapidement afin que les utilisateurs puissent avoir un meilleur service plus rapidement.~~

## 4.4 Vue architecturale (à l’extérieur de la classe)

### 4.4.1 Diagramme





### 4.4.2 Diagramme de séquence

Afin de d’accélérer le processus d’installation des guichets, il faudra définir des scripts qui seront exécutés lors de l’installation. Afin de documenter le processus, il sera nécessaire de créer deux diagrammes de séquences afin de montrer la logique derrière le script.

Le premier diagramme de séquence va définir les actions sur le guichet automatique afin d’avoir une automatisation de l’installation du guichet. Pour ce faire, il trouvera son adresse IP et va l'enregistrer dans le guichet, ensuite, il va mettre à jour la table de routage et les différents éléments du réseau afin qu’ils sachent que le réseau existe. Finalement, il enverra une notification au système central de la banque afin qu’il exécute le script pour ce guichet.

Le deuxième diagramme de séquence va définir les actions prises par le système central de la banque afin de mettre à jour la table contenant les information pour les différents guichets. Il va aussi fournir les clés de sécurité afin que le guichet et le système central communiquent sécuritairement.

Ces deux diagrammes seront ensuite donnés à un spécialiste BASH afin que celui-ci les réalise et sache exactement quoi tester.

### 4.4.3 Légende

### 4.4.4 Texte de description du diagramme

Les vues de déploiements, d’activité et de séquence permettent de représenter globalement chaque aspect important de l’ajout de guichet automatique en masse au réseau. Les trois vues permettent d’avoir une vue d’ensemble quant aux étapes à suivre pour assurer l’ajout de nos trois tactiques afin de résoudre les problèmes de modifiabilité lié à l’ajout de guichet automatique au réseau. L’ajout sera ainsi rapide, fiable et demandant très peu de changement.

### 4.4.5 Table de description des éléments du diagramme

Diagramme de déploiement

| Élément | Description |
| --- | --- |
| Guichet Bancaire | Représentation de la machine physique ainsi que de ses composants physiques principaux. |
| Routeur | Matériel réseau permettant la communication entre le guichet et le serveur de la banque. |
| Serveur | Serveur principal de la banque. |
| Base de données | Emplacement de stockage des informations des guichets. |

### 

Diagramme d’activité

| Élément | Description |
| --- | --- |
| Guichet | Élément physique étant ajouté au réseau. |
| Réseau | Représentation du réseau ainsi que de la communication TCP/IP utilisé pour transférer l’information entre le guichet et le serveur. |
| Serveur | Élément physique principal de la banque. |
| Lignes pleines | Redirection vers la prochaine action à exécuter |
| Point d’entrée | Début de la séquence d’activité |
| Point de sortie | Fin possible de la séquence d’activité(selon un cas non respecté ou la fin de la séquence d’action). |

### 4.4.6 Texte décrivant la relation entre les éléments et les tactiques

Le fait d’avoir de l'encapsulation de la sécurité permet au Système d’être déployé plus facilement et d’avoir des machine qui entre sur le réseau à un Rythme satisfaisant. Le fait d’avoir un abstract common service permet aussi de déployer avec plus de flexibilité. Finalement, pour mettre à jour le système, nous allons utiliser un script pour mettre à jour les routeurs, serveurs et autre items sur le réseau.

# 5 Scénario :

| **objectifs d'affaires** | Permettre la modification de l'interface utilisateur affichée dans l’ATM, avec l'objectif d'adapter l'interface de l’ATM aux améliorations graphiques. |
| --- | --- |
| **Source** | Développeur |
| **Stimulus** | Mise en marche de l'ATM |
| **Artéfact** | Code de la vue de l'architecture MVC |
| **Environnement** | Mode d'opération normal |
| **Réponse** | L'ATM affiche la nouvelle interface graphique |
| **Mesure de la réponse** | Il est possible de mettre à jour l'interface graphique de l’ATM sans affecter les modules physiques de l'ATM et la logique d'affaires de la banque. |
| **Questions** | 1. Quelles couleurs afficher dans l'interface graphique? 2. Quelle est l'interface graphique de base? 3. L'interface graphique est disponible dans quelles langues? |

## 5.1 Tactique 1 : Module divisé

**Description**: Le système est composé de plusieurs modules divisés.

**Justification**: permets de réduire les coûts de modifications des modules, comme l'interface graphique qui peut être mise à jour sans affecter les autres composants du système.

## 5.2 Tactique 2 : Encapsulation

**Description**: Le code de l'interface graphique est encapsulé et isolé des autres modules du système.

**Justification**: En encapsulant le code faisant partie de l'interface graphique, il permettra de modifier la vue de l'architecture MVC sans affecter les autres composants du système.

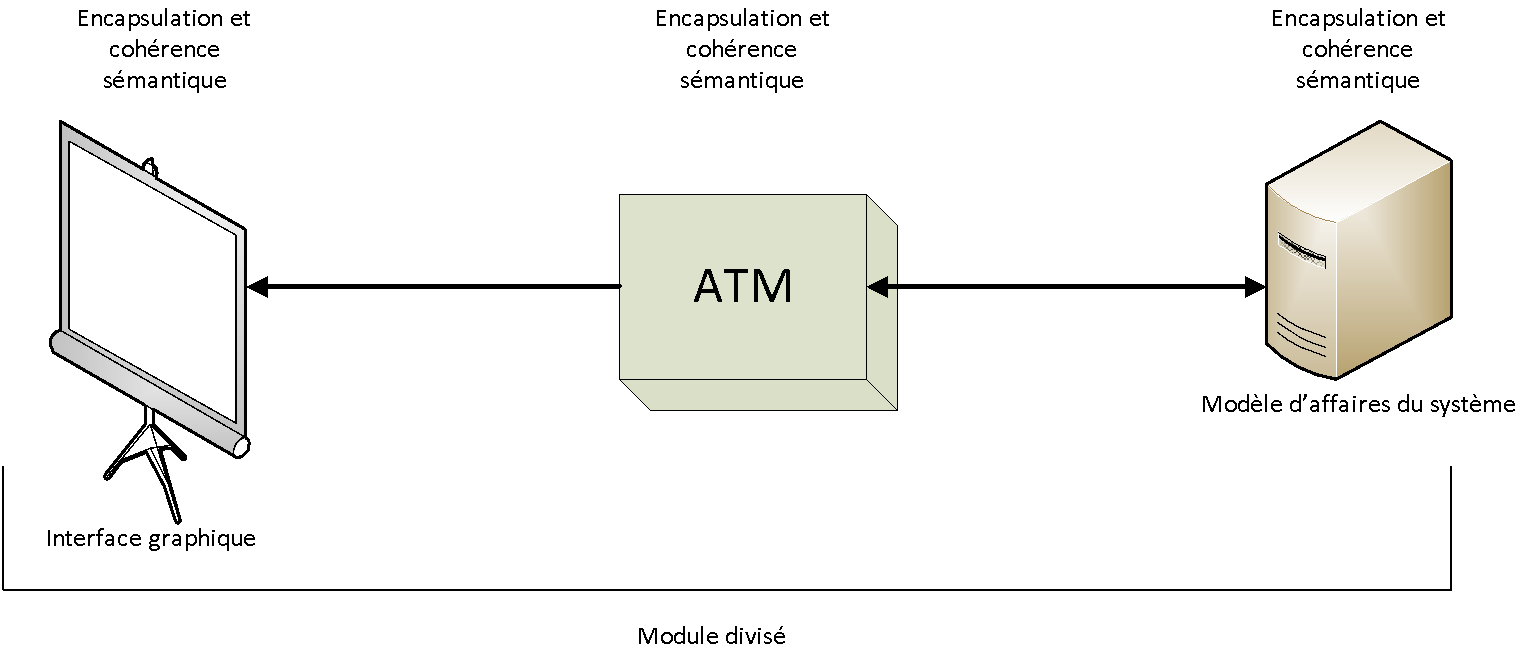
## 5.3 Tactique 3 : Augmenter la cohérence sémantique

**Description**: faciliter la séparation de la responsabilité d'affichage à l'interface graphique des autres composants du système.

**Justification**: En utilisant cette tactique, il sera plus facile de séparer les responsabilités d'affichages graphiques, de lecture de cartes bancaires et du modèle d'affaires du système.

## 

## 5.4 Diagramme 1.0 Modélisation de la modificabilité du système d'ATM.



### 5.4.1 Description du diagramme:

Trois composants du système, l'interface graphique, l’ATM et la Banque ont des relations de modificabilité en cas de problèmes. L'interface graphique respectera les tactiques de module divisé, d'encapsulation et d'augmentation de la cohérence sémantique, au moment de son implémentation dans le système.

### 5.4.2 Table de description des éléments du diagramme :

| **Tactiques** | **Descriptions** |
| --- | --- |
| Module divisé | Diviser les modules du système |
| Encapsulation | Isolation de modules |
| Augmenter la cohérence sémantique | Définitions efficaces de responsabilités des modules |

### 5.4.3 Relations :

Les cercles représentent l'encapsulation et l'augmentation de la cohérence sémantique.

La ligne en bas représente la tactique de modules divisés.

# 6 Scénario : Modifiabilité

| **objectifs d'affaires** | Permettre d’ajouter un module qui permet à l’utilisateur de virer de l’argent de son compte vers le compte d’une autre personne |
| --- | --- |
| **Source** | Développeur |
| **Stimulus** | Ajout de la fonctionnalité |
| **Artéfact** | Code |
| **Environnement** | Environnement de développement |
| **Réponse** | Tâche à réaliser pour ajouter une fonctionnalité |
| **Mesure de la réponse** | La fonctionnalité doit être déployée d’ici deux mois |
| **Questions** | 1. Doit-il y avoir une particularité à l’écran pour mentionner à l’utilisateur qu’il vire de l’argent dans un compte qui n’est pas le sien (avertissement, couleur différente, etc.)? 2. Quelles doivent être les informations demandées à l’écran pour faire le virement? 3. Est-ce que tous les GAB doivent être en mesure de supporter cette nouvelle fonctionnalité? 4. La fonctionnalité doit-elle être construite pour être en mesure d’être modifiable dans le futur? 5. Doit-on ajouter un menu complet pour un virement ou le chemin pour arriver à la fonction se retrouve en sous-menu dans la fonction “transfert” déjà implémentée? Est-elle une fonction à part ou une amélioration de la fonction de transfert? |

## 6.1 Tactique 1 :

**Description**: Séparer les modules de type de transaction

**Justification**: Afin de pouvoir les gérer indépendamment (faire un ajout, une modification ou supprimer) sans en impacter les autres, il est primordial que chacune des transactions proposés dans le GAB soient des modules indépendants qui héritent d’un module Transaction, regroupant les classes communes. Ainsi, on améliore la modifiabilité en temps de développement si un changement dans le code du logiciel s’impose, mais aussi la cohésion des modules en question, tout en réduisant le couplage.

## 6.2 Vue architecturale (à l’extérieur de la classe)

### 

Le diagramme ci-dessus représente une vue “utilise”, de la famille modules. Elle est nécessaire ici pour montrer le changement de l’organisation des modules du logiciel du GAB dans lequel on ajoute la nouvelle fonctionnalité de virement qui, comme les autres, représente un type de transaction. C’est pourquoi chacune d’entre elles se lie au module de la transaction afin d’en hériter des attributs ou méthodes communes.

Cette vue architecturale fait directement référence à l’attribut de modifiabilité puisqu’elle donne une image plus technique et détaillée de l’architecture du code du logiciel. Celle-ci est donc utile pour un aperçu des éléments et des relations qui font référence à des procédés de développement.

# 7 Scénario : Modifiabilité

| **Objectifs d'affaires** | Ajout de partenariat avec d’autres banques, notamment un support pour les transactions avec celles-ci. |
| --- | --- |
| **Source** | L’équipe de développement reçoit une demande des gestionnaires qui veulent effectuer le partenariat |
| **Stimulus** | Demande d’ajout de modules spécifiques aux nouvelles banques. |
| **Artéfact** | Code et interface. |
| **Environnement** | Compile time, build time, design time. |
| **Réponse** | Ajouter les modifications.  Tester les modifications.  Déployer les modifications. |
| **Mesure de la réponse** | Le temps d’ajout du module.  La complexité des documents des banques.  Le coût d’ajout des modifications.  Le risque des changements des traitements de requêtes.  La grosseur des modules à ajouter. |
| **Questions** | 1. À quel point les architectures des banques sont-elles différentes ? 2. Comment faire la requête de message à la banque ? 3. Quelle est la criticité des différents systèmes à connecter ? 4. À quel point est-il difficile de mettre en contact les équipes de développement concernées ? 5. Est-ce que les banques concernées possèdent déjà une API? |

## 7.1 Tactique 1 : Reduce size of a module - Split module

**Description**: Diviser les modules pour réduire les dépendances et les coûts de changements. Création de modules pour chaque banque indépendamment.

**Justification**: Permettre l’ajout d’un support pour de nouvelles banques facilement, car chacune d’elles va avoir sa propre implémentation et sa propre logique de traitement. Ainsi, il sera plus rapide d’identifier où faire les modifications.

## 7.2 Tactique 2 : Reduce coupling - Encapsulate

**Description**: Utilisation d’API des autres banques si disponible et .

**Justification**: L’utilisation de l’API des banques permet de faire les modifications plus rapidement et de donner une certaine documentation pour exécuter les changements. L’API va permettre d’utiliser une technique uniforme propre à la banque. Notre API permettra aux banques de faire de même avec notre système.

## 7.3 Tactique 3 : Reduce coupling - Use an intermediary

**Description**: Le patron façade, en tant qu’intermédiaire, sert d’interface pour les fonctionnalités internes de notre banque, que les banques partenaires peuvent utiliser.

**Justification**: Le couplage est fortement réduit grâce à une façade puisque les banques partenaires n’ont pas à adapter leur code à toutes nos fonctionnalités; elles passent simplement par l’interface unique qui à son tour passe les informations aux bons modules internes.

## 7.4 Vue architecturale (à l’extérieur de la classe)

### 7.4.1 Diagramme



**Figure 1.0 :** Diagramme du scénario de modifiabilité.

- Le diagramme illustré dans la figure 1.0 consiste à fournir une stratégie pour améliorer la modifiabilité du système, notamment lors de l’ajout de partenariats avec d’autres banques. Tout d’abord, nous introduisons une façade à notre système afin de faire abstraction aux différents modules internes séparés en utilisant la tactique Split Module. Aussi, l’encapsulation d’APIs dans chaque système indépendant rend le fonctionnement interne de ceux-ci non-affectés par tout changement aux interactions entre les banques, ce qui est un gros avantage pour la modifiabilité.

**Table de description des éléments du diagramme**

| **Élément** | **Responsabilité** |
| --- | --- |
| Façade/API | Fournir les fonctionnalités nécessaires aux banques |
| Modules (Transaction, UI, Autres) | Ensemble de modules séparées qui contiennent les fonctionnalités pour une action spécifique |
| Banques API | Interface fournie par la banque sous forme d’une API |
| Système Banque X | Système global de la banque qui englobe les différents modules et logiciels. |

**Table de description des relations des tactiques entre ces éléments**

| **Élément** | **Description** |
| --- | --- |
| Split Module | Séparation des modules en sous-modules pour réduire les coûts de modification. |
| Intermediary | Détruire les dépendances entre les modules internes et les autres systèmes bancaires. |
| Encapsulate | Encapsulation des services publics ou accessibles aux systèmes externes. |

**Description des relations entre les éléments**

En utilisant la tactique split module, on permet de réduire les coûts de modifications ainsi qu’une meilleure utilisabilité. L’encapsulation des services accessibles aux systèmes externes permet l’abstraction du fonctionnement interne spécifique à chaque banque. Finalement, en utilisant la tactique Intermediary, le couplage est réduit et les modules passent par une façade qui s’occupe de bien faire le “routing” des données.

# 8 Scénario :

| **objectifs d'affaires** | Faciliter le changement des composantes matérielles de l’ATM afin de permettre l’évolution de l’ATM ou le changement de fournisseurs. |
| --- | --- |
| **Source** | Développeur |
| **Stimulus** | Réception d’une requête de changement |
| **Artéfact** | Code |
| **Environnement** | Conception de l’ATM |
| **Réponse** | Découpler le code de l’ATM des composantes matérielles |
| **Mesure de la réponse** | Il est possible d’interchanger les composantes de l’ATM sans avoir à changer une ligne de code. |
| **Questions** | 1. Quelles sont les interfaces standards pour communiquer avec un lecteur de cartes? 2. Est-ce que les composantes ont aussi une interface physique compatible? |

## 8.1 Tactique 1 : Encapsulation

**Description**: Isolation du code faisant interface avec les composantes matérielles.

**Justification**: En isolant le code faisant interface avec les composantes matérielles, il sera plus facile d’interchanger une composante puisqu’il n’y aura aucun effet sur le reste du code. Si le code faisant interface avec le matériel fait appel à des méthodes et API standards, il n’y aura aucune modification à faire.

## 8.2 Tactique 2 : Restriction des dépendances

**Description**: Restreindre les dépendances entre les différentes classes pour réduire le couplage permettant ainsi une utilisation plus souple du système et de ses composantes.

**Justification**: En utilisant cette tactique il sera plus facile de changer une composante puisque ce changement aura seulement un impact sur une infime partie du code.

## 8.3 Tactique 3 : Adaptateur

**Description**: Une évaluation des efforts, coût et maintenant est nécessaire pour le choix de l'adaptateur.

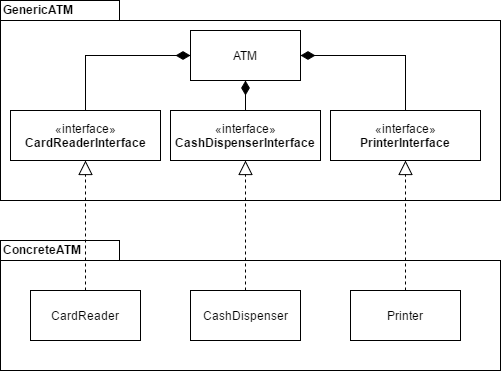
1. Le fournisseur doit fournir une interface adaptable au système existant.
2. Le système existant doit s’adapter à la nouvelle interface des fournisseurs

**Justification**:

1. Exiger au fournisseur une interface adaptable permettrait de réduire l’effort et la maintenance de l’interface.
2. Un système adaptable à toute interface augmenterait considérablement la cohésion de celui-ci.

## 8.4 Vue architecturale (à l’extérieur de la classe)

### 8.4.1 Diagramme



### 8.4.2 Légende



### 8.4.3 Texte de description du diagramme

Diagramme UML démontrant les interfaces utilisables pour les composantes physiques de l’ATM

### 8.4.4 Table de description des éléments du diagramme

| Élément | Description |
| --- | --- |
| ATM | Classe principale de l’ATM qui contiendra les instances des interfaces génériques implémentés dans par les classes du package ConcreteATM |
| Packages | * ConcreteATM : Contient les classes externes au logiciel de l’ATM. Principalement les composantes physiques. * GenericATM : Contient le controlleur de l’ATM et les interfaces qui seront implémentés par les composants physiques de l’ATM |
| Interfaces | Les interfaces contiennent les signatures des méthodes nécessaires au bon fonctionnement de l’ATM. Exemple : l’interface card reader contiendra les méthodes readCard() et ejectCard(). |
| Classes | Manifestation concrète des interfaces. Elle implémentent les méthodes définies dans les interfaces en fonction des composantes physiques. |

### 8.4.5 Texte décrivant la relation entre les éléments et les tactiques

Le diagramme démontre à la fois la tactique 1 et la tactique 2. Tout d’abord, il démontre l’encapsulation puisqu’il contient des interfaces qui isole le code du système interne de l’ATM tout en permettant l’implémentation de certaines méthodes dans le package ATMConcrete. De plus, une implémentation de la sorte réduit le couplage et rend les deux packages complètement indépendants ce qui comble notre deuxième tactique étant la restriction des dépendances.

# 9 Scénario :

| **objectifs d'affaires** | Le guichet devrait pouvoir accepter et distribuer plus d’un type de monnaie (Ex: USD et CAD) |
| --- | --- |
| **Source** | L’équipe de développement logiciel, et l’équipe de techniciens |
| **Stimulus** | Il s’agit d’une modification des capacités du guichet autant au niveau logiciel que matériel |
| **Artéfact** | Des modifications seront nécessaires au niveau logiciel dans les interfaces utilisateur et d’administration, mais aussi au niveau du code du guichet et des interactions avec les serveurs  Au niveau matériel, le guichet devra pouvoir stocker séparément les billets de monnaies différentes et les distribuer par la même fente. |
| **Environnement** | Les changements affectent le guichet au moment de sa conception et de son exécution. |
| **Réponse** | Implémenter, Tester et déployer la modification. |
| **Mesure de la réponse** | * Artéfacts : Peu complexe, mais affecte un grand nombre de composantes * Effort : Moyen à Élevé, selon le fonctionnement actuel * Calendrier : Logiciel → Temps moyen, complexité faible;   + Matériel → Long car ne pourra pas être fait sur place * Argent : Coût d’opportunité élevé (Lieux touristiques et aéroports principalement) * Peut affecter la disponibilité, car il s’agit d’un service supplémentaire (taux de change) * De nouveaux types de défauts seront introduits dans le système |
| **Questions** | 1. Dans quelle ampleur le matériel doit-il être modifié pour stocker deux types de monnaies 2. Le distributeur de billet pourra-t-il supporter des billets de différentes dimensions? |

## 9.1 Tactique 1 : Encapsulation

**Description**: Réduire la probabilité qu’un changement dans un module se propage vers un autre module. Une interface limite les interactions avec ce module et cache ses détails d’implémentation.

**Justification**: Une interface devra être créée pour définir les fonctions à implémenter par chaque monnaie. Partout dans le code, nous allons pouvoir maintenant programmer contre cette interface au lieu d’une forte dépendance avec une monnaie précise.

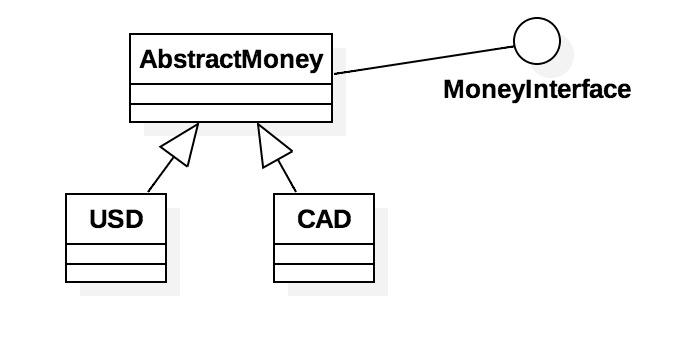
## 9.2 Tactique 2 : Abstract Common Services

**Description**: Réduire les coûts de modification en regroupant et en implémentant une seule fois, de manière plus générique, une fonctionnalité ou service similaire à plusieurs modules.

**Justification**: Une classe abstraite devra être créée pour les opérations monétaires. Chaque monnaie devra étendre cette classe abstraite afin d’implémenter ses particularités telles les montants des coupures, les limites de dépôt et de retrait, et autres. La classe abstraite permettra de réutiliser les services communs à plusieurs monnaies.

## 9.3 Vue architecturale (à l’extérieur de la classe)

### 9.3.1 Diagramme



### 9.3.2 Légende

### 9.3.3 La classe abstraite AbstractMoney implémente l’interface MoneyInterface

* Les classes USD et CAD étendent la classe abstraite *AbstractMoney*

### 9.3.4 Texte de description du diagramme

Il faudra créer une classe abstraite nommée AbstractMoney qui implémentera les fonctionnalités communes à la majorité des monnaies. Chaque monnaie étendra cette classe et pourra définir ses particularités (nombre de décimales, symbole, coupures, etc). La classe AbstractMoney implémentera aussi l’interface MoneyInterface, qui définira les services à offrir au système de l’ATM pour chaque monnaie. Le reste du système sera modifié pour utiliser cette interface.

### 9.3.6 Table de description des éléments du diagramme

| **Élément** | **Description** |
| --- | --- |
| MoneyInterface | Interface utilisée par le reste du système de l’ATM |
| AbstractMoney | Classe abstraite implémentant de façon générale les éléments communs à la plupart des monnaies |
| USD, CAD, autres | Implémentations de monnaies précises, avec leur propres particularités. |

### 9.3.7 Texte décrivant la relation entre les éléments et les tactiques

**Tactique 1 : Encapsulation**

L’interface *MoneyInterface* cache les détails d’implémentation des monnaies, et garantit un service aux autres composants de l’ATM. L’impact de l’ajout d’autres monnaies sera donc minimisé.

**Tactique 2 : Abstract Common Services**

La classe abstraite *AbstractMoney* implémentera, de manière générique, les éléments communs à la majorité des monnaies. Par exemple, elle comprendrait une méthode permettant d’afficher un montant d’argent, tout en laissant le soin à l’instance de la monnaie de définir son symbole, le nombre de décimales, etc.