LOG430 – Architecture logicielle

# Scénarios de qualité, tactiques et justifications des attributs produits par les équipes en classe

Groupe 01

# < Modificabilité > – Justification (en classe)

On veut que l’ATM puisse être modifié pour accepter de nouveaux standards de cartes bancaires, possiblement de nouvelles institutions bancaires, par exemple des banques d’autres pays. La banque en tire avantage car elle peut percevoir plus de frais d’utilisations avec ces nouvelles cartes. Le client n’a pas la frustation de se voir refuser sa carte.

# <Modificabilité> – Scénario 1 (en classe)

**Source**: L’administrateur de la banque.

**Stimuli**: Requête de supporter une nouvelle banque ou standard de carte bancaire.

**Artéfact**: Code du module d’acceptation de cartes.

**Environnement**: Design pour supporter facilement de nouveaux standards, on doit modifier le code (recompiler).

**Réponse**: Le code qui permet de lire les cartes a été modifié pour supporter le nouveau standard, il a été testé et déployé sur les machines ATM. Le patron adapter permet d’ajouter la fonctionnalité en ajoutant une seule classe.

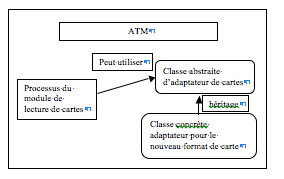
**Mesure de la réponse**: Temps nécessaire pour supporter la nouvelle carte.

# <Modificabilité> – Tactique 1 (en classe)

**Description**: Reduce coupling – Abstract common service

**Justification**: Selon le patron adaptateur, la logique commune à tous les types de cartes bancaires sera définie une seule fois dans la classe abstraite, et les classes concrètes contiendront les spécificités pour chaque format de carte bancaire.

# <Modificabilité> – Vue Architecturale (en classe)



Groupe 02

Scénarios de qualité, tactiques et justifications des attributs produits parles équipes en classe

<Modifiabilité> – Justification

Une personne daltonienne, vient faire un retrait mais l’interface usagé est aux couleurs de la banque (rouge et bleu). Ces couleurs n’étant pas perceptibles par ce client, celui-ci a de fortes difficultés à lire et donc à effectuer son retrait.

<Modifiabilité> – Scénario

**Source** : Utilisateurs daltoniens (environ 10% de la population mâle)

**Stimuli** : Les daltoniens veulent pouvoir comprendre l’interface correctement sans difficultés

**Artéfact** : Interface utilisateur

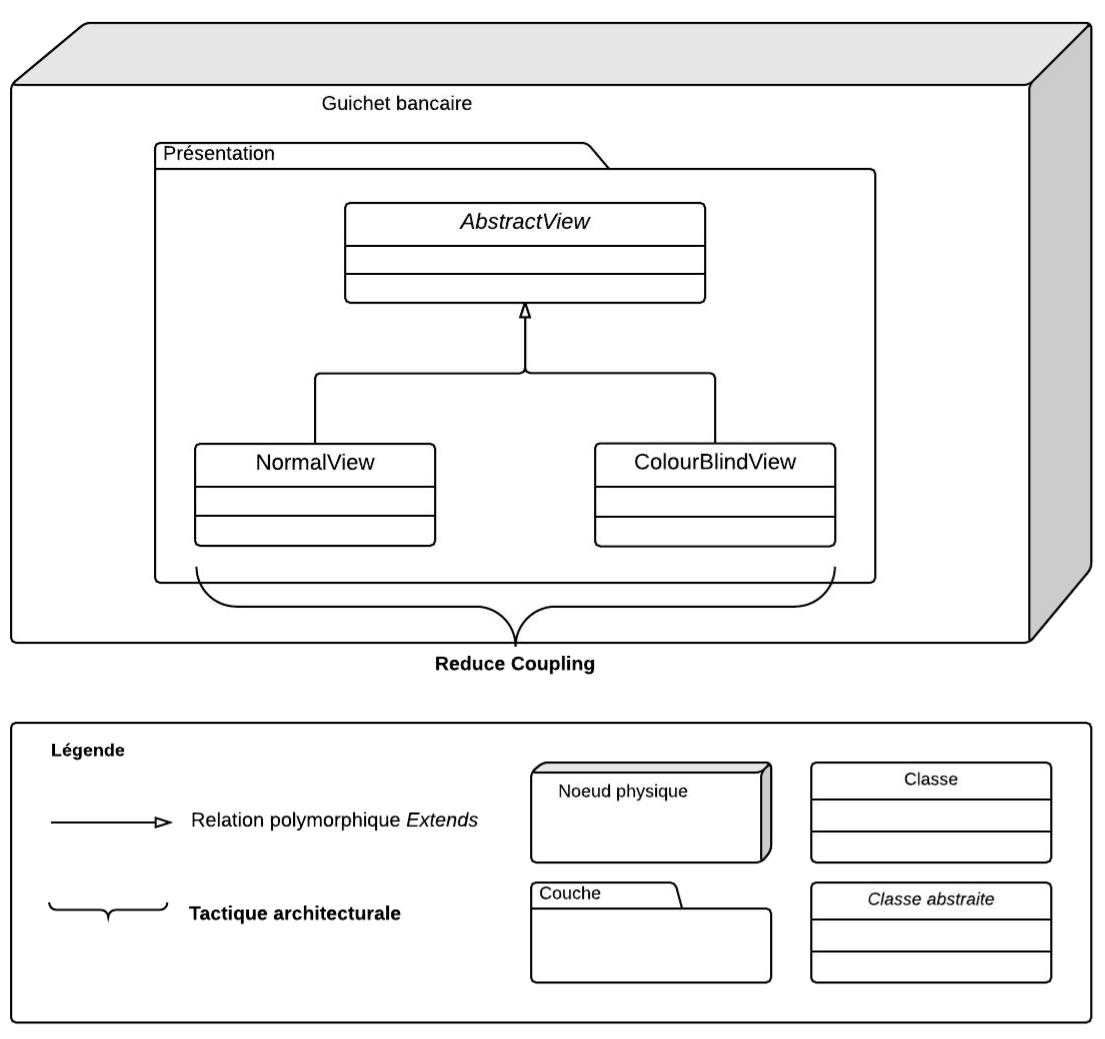
**Environnement** : Production

**Réponse** : Ajouter un bouton dans l’interface utilisateur qui change le rouge et le vert dans l’interface pour d’autres couleurs. (Mode daltonien)

**Mesure de la réponse** :

* ●  Mesurer le temps moyen de la transaction d’une personne daltonienne
* ●  Compter la fréquence d’utilisation du bouton pour le mode daltonien
* <Modifiabilité> – Tactique
* **Description** : Reduce Coupling
* **Justification** : On encapsule la vue du guichet afin de facilement pouvoir passer d’une vue à l’autre. L’utilisateur aura le choix entre une vue normal ou une vue adaptée aux daltoniens.

<Modifiabilité> – Vue Architecturale



La vue architecturale ci-dessus nous montre qu’une classe abstraite est utilisée pour la vue de l’interface utilisateur du guichet bancaire. Cette classe contient l’essentiel de la fonctionnalité d’une vue. Ensuite, par polymorphisme, une vue normale et une vue pour les daltoniens héritent de cette classe en changeant notamment les couleurs.

Éléments du diagramme

|  |  |
| --- | --- |
| **Élément** | **Description** |
| Guichet | Le noeud physique correspondant à la machine de guichet automatique bancaire et le logiciel qui y est exécuté. |
| Guichet bancaire / Présentation | La couche Présentation du guichet bancaire comporte les éléments d’interface graphique du guichet. Cette couche implémente la tactique *Reduce Coupling* pour augmenter la modifiabilité. |
| AbstractView | La vue abstraite contenant les fonctionnalités de base d’une vue de guichet bancaire. |
| NormalView | La vue concrète contenant les fonctionnalités d’une vue normale (pas pour les daltoniens). |
| ColourBlindView | La vue concrète contenant les fonctionnalités d’une vue pour les daltoniens. Les couleurs sont différentes. |

La tactique *Reduce Coupling* est ici implémentée en utilisant AbstractView partout où la vue normale était utilisée auparavant et en surchargeant seulement les éléments nécessaires dans NormalView et ColourBlindView. On réduit ainsi le couplage entre toutes les classes collaboratrices de NormalView en les faisant utiliser AbstractView qui est de plus haut niveau et moins spécifique.

Cette tactique comble le besoin de satisfaire les utilisateurs daltoniens tel que décrit dans le scénario.

Groupe 03

**Scénarios de qualité, tactiques et justifications des attributs produits par les équipes en classe**

**Équipe 3 < Modificabilité > – Justification (en classe)**

Offrir la possibilité d’effectuer un changement de configuration du système afin de par exemple permettre de modifier la liste des serveurs avec lesquels le serveur doit communiquer.

**<Modificabilité> – Scénario 1 (en classe)**

**Source**: Technicien affecté à la maintenance du guichet.

**Stimuli**: Modifier la configuration du système contenant par exemple la liste des adresses pour rejoindre les serveurs.

**Artéfact**: Processus gérant les modifications de configuration de la machine GAB.

**Environnement**: Environnement de maintenance

**Réponse**: Effectuer la modification, Tester la modification. Déployer la modification

**Mesure de la réponse**: Effort en temps requis pour modifier la configuration sur les machines. Impact des modifications effectuées sur le fonctionnement du système.

**<Modificabilité> – Tactique 1 (en classe)**

**Description**: *Defer binding*

**Justification**: Implémenter dans la configuration les variables de fonctionnement les plus probables de changer telles que le montant maximal des retraits ou les adresses des serveurs dans le but d’accélérer le processus de modification si un changement devient nécessaire.

**<Modificabilité> – Vue Architecturale (en classe)**

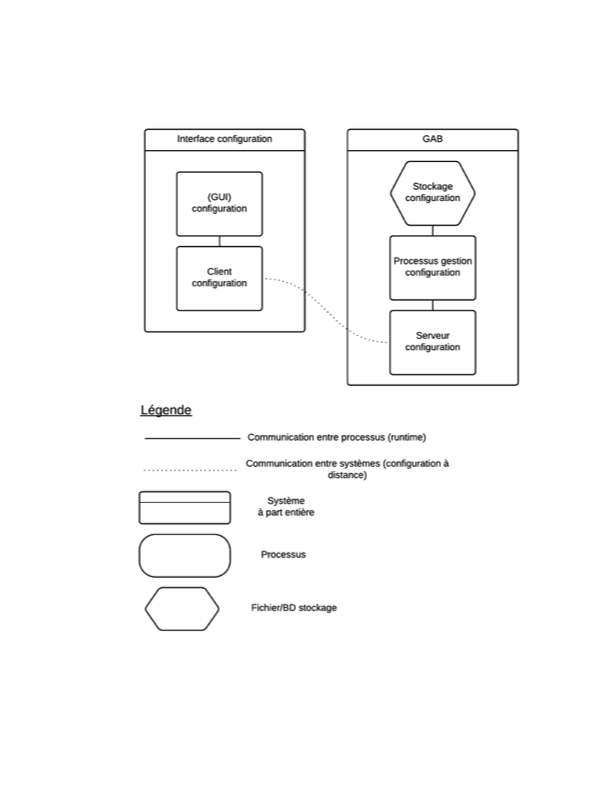


Diagramme du système permettant la configuration du GAB à distance par un technicien. Deux systèmes sont présents soit l’interface de configuration pouvant être utilisée sur une station indépendante pour tous les GAB. Le technicien utilise l’interface graphique pour accéder au GAB et également modifier la configuration de celui­ci.

**Table de description des éléments du diagramme**

|  |  |
| --- | --- |
| **Élément** | **Description** |
| Interface configuration | Solution logicielle permettant de se connecter au GAB et modifier la configuration de celui­ci. |
| GAB | Solution logicielle implémentée dans le guichet automatique bancaire |
| (GUI) Configuration | Processus de l’interface utilisateur présentée au technicien pour la configuration |
| Client configuration | Processus client pour récupérer et transmettre les configurations au GAB. |
| Serveur configuration | Processus du GAB permettant au client configuration de se connecter pour récupérer et modifier les configurations. |
| Processus gestion configuration | Processus gérant l’enregistrement des configurations dans le fichier de configuration interne du GAB. |
| Stockage configuration | Fichier ou BD contenant les variables de configuration applicables au GAB. |

En offrant une interface de configuration externe on permet au technicien de se connecter à divers GAB à partir du même point d’accès. Étant donné que nous offrons un espace de stockage pour les configurations il est possible de stocker les variables de configuration dans ce­dernier et donc de permettre la modification des paramètres d'exécution une fois que le guichet est en production.

La tactique employée permet le stockage de configurations dans chaque GAB et la modification aisée de celles­ci sans nécessairement devoir envoyer un technicien sur place pour chaque modification. Puisque les configurations sont stockées dans un fichier modifiable il est possible d’effectuer des modifications lorsque le GAB est en production ce qui élimine des coûts importants de maintenance. Ceci est dû au fait que l’on permet de modifier un grand nombre de variables d’exécution sans devoir modifier la solution logicielle elle­même.

Groupe 05

**< Modificabilité > – Justification (en classe)**

Séparation des modules dépôt et retrait

**<Modificabilité> – Scénario 1 (en classe)**

**Source**: Utilisateur du guichet

**Stimuli**: L'utilisateur ne peut pas déposer ses billets hors d'une enveloppe

**Artéfact**: Modifier le code et un composant dans le guichet

**Environnement**: Opération normale

**Réponse**: Faire les modification, les tester et déployer la solution

**Mesure de la réponse**: Équipe de 3 personnes pour la séparation des modules; 1 personne pour les modifications matérielles et 2 pour l'implémentation du code. En 2 mois jusqu'au déploiement.

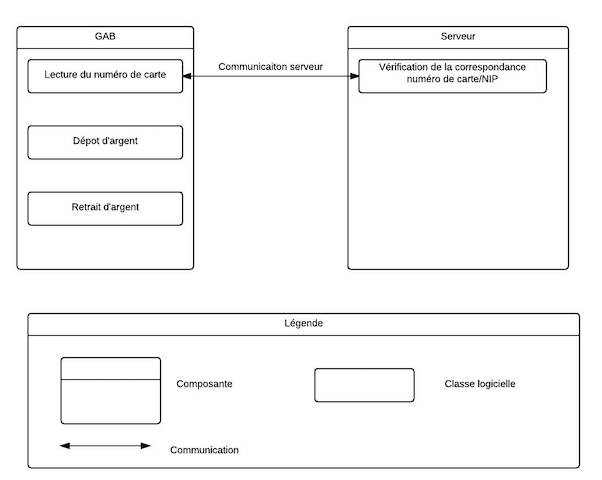
**<Modificabilité> – Tactique 1 (en classe)**

**Description**: Séparation des modules dépôt et retrait

**Justification**: On sépare les modules de façon à réduire le couplage entre eux et le reste de l'application lors de la modification de celui de dépôt.

**<Modificabilité> – Vue Architecturale (en classe)**

* Diagramme:



Texte de description du diagramme

* Le diagramme suivant représente l’interaction qu’il y a entre un guichet automatique bancaire et un serveur. On peut remarquer que le guichet bancaire s’occupe de lire le numéro de carte qui est immédiatement envoyé au serveur pour vérification. Le serveur vérifie le NIP et rend son verdique sur la validité de celui-ci. Ensuite, l’utilisateur peut faire des dépôts, retraits,etc.

Table de description des éléments du diagramme

* **GAB**: Guichet automatique bancaire que le client utilise pour effectuer un retrait.
* **Serveur**: Serveur de la banque qui gère toute l’information des clients.
* **Lecture du numéro de carte**: Classe logicielle qui gère toute la lecture de la carte.
* **Dépôt d’argent**: Classe logicielle qui s’occupe de réaliser des dépôts au guichet automatique.
* **Retrait d’argent**: Classe logicielle qui s’occupe de réaliser des retraits au guichet automatique.
* **Vérification de la correspondance numéro de carte/NIP**: Classe logicielle du serveur qui vérifie la validité du NIP/numéro de carte pour ainsi effectuer des opérations.

Texte décrivant la relation entre les éléments et la tactique

À l’intérieur du composant «GAB», nous avons séparé les classes de dépôt et de retrait. Ceci permet de réduire le couplage entre eux.

Groupe 06

**< Modificabilité > – Justification (en classe)**

L’équipe de marketing désire avoir une sorte de Guichet automatique dans lequel il est possible de seulement faire des dépôts dans le but de réduire l’achalandage au guichet et ainsi faciliter les dépôts dans leur banque.

**<Modificabilité> – Scénario 1 (en classe)**

**Source**: L’équipe de marketing.

**Stimuli**: Création de guichets permettant seulement les dépôts.

**Artéfact**: Code

**Environnement**: Compile time, on n’ajoutera pas toutes les fonctionnalités aux guichets permettant seulement les dépôts.

**Réponse**:

* ●  Tester la modification
* ●  Déployer la modification
* **Mesure de la réponse**: 8 semaines jusqu’au déploiement sur les nouveaux guichets automatiques
* **<Modificabilité> – Tactique 1 (en classe)**
* **Description**: Reduce size of a Module; Split module
* **Justification**: Le fait de bien séparer les fonctionnalités va simplifier les modifications futurs et en réduire les coûts. De plus, il va être plus facile, lors de la compilation, de compiler pour le bon type de guichets.

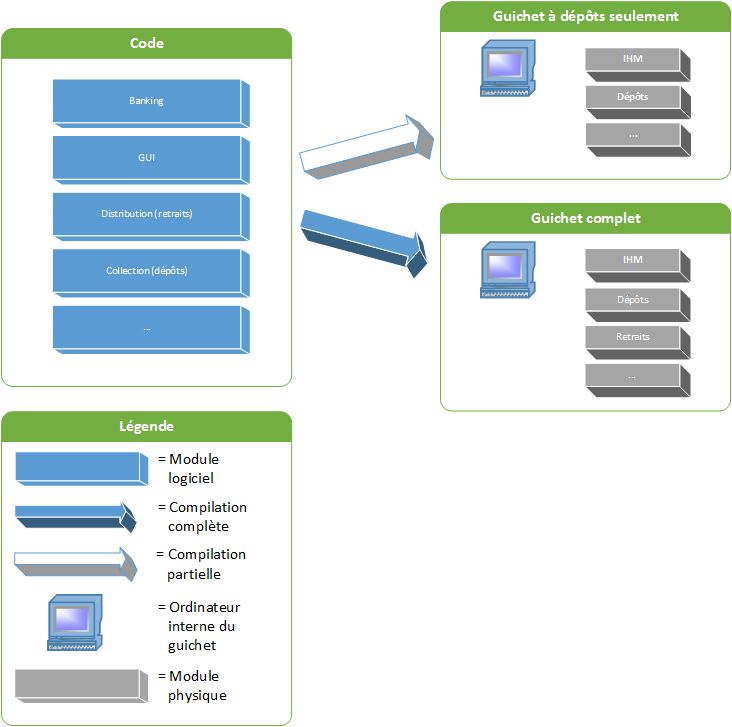
**<Modificabilité> – Vue Architecturale (en classe)**

Diagramme exprimant la séparation par module de l’application afin d’offrir la possibilité de moduler le code selon les besoins du produit.

**Relation entre les éléments et la tactique**

Dans ce diagramme, il est possible de l’implémentation de la tactique de réduction de la grosseur des modules. Chacune des fonctionnalités du guichet est séparée en un module spécifique à sa tâche. Ensuite, lors de la compilation du code pour un certain type de guichet, des conditions arguments peuvent être entrés afin de ne compiler que partiellement le code afin d’avoir accès aux modules qui seront disponibles dans le futur guichet.

Cette tactique permettra de n’avoir qu’à maintenir un seul projet pour chaque type de guichet qui sera ajouté dans le futur.



**Réponse au scénario de qualité**

La séparation en modules du code permet de répondre au scénario de qualité, car la création de divers types de guichet sera physique ne demandera que d’ajouter ou de retirer les modules nécessaires au fonctionnement de celui­ci.

|  |  |
| --- | --- |
| **Élément** | **Description** |
| Code | Code unique du guichet. |
| Guichet à dépôts seulement | Module physique du guichet de l’entreprise n’ayant que la fonctionnalité de dépôt de disponible. |
| Guichet complet | Module physique du guichet de l’entreprise possédant les fonctionnalités complètes (par défaut) disponibles. |
| Ordinateur interne du guichet | Module logique central du guichet. |
| Module logiciel | Bloc module à fonctionnalité spécifique |
| Module physique | Pièce physique du guichet bancaire. |

Groupe 07

**< Modificabilité > – Justification (en classe)**

Un système bancaire GAB est un système qui est souvent porter a être modifié. Donc, la modificabilité est importante, puisqu’il est presque certain que le système va changer au cours de sa durée de vie. Si on prévoit un grand nombre de changement, il peut être avantageux et moins coûteux de mettre en place des mécaniques permettant de modifier plus facilement le logiciel.

**<Modificabilité> – Scénario ­ Langue système (en classe)**

**Source**: L’utilisateur désire utiliser le système dans différentes langues

**Stimuli**: Modifier l’affichage du langage dans le système

**Artéfact**: Des fichiers de ressources de langue qu’on accède selon un paramètre de langue du système

**Environnement**: Au “runtime” : la langue peut être modifiée lors de l’exécution du système

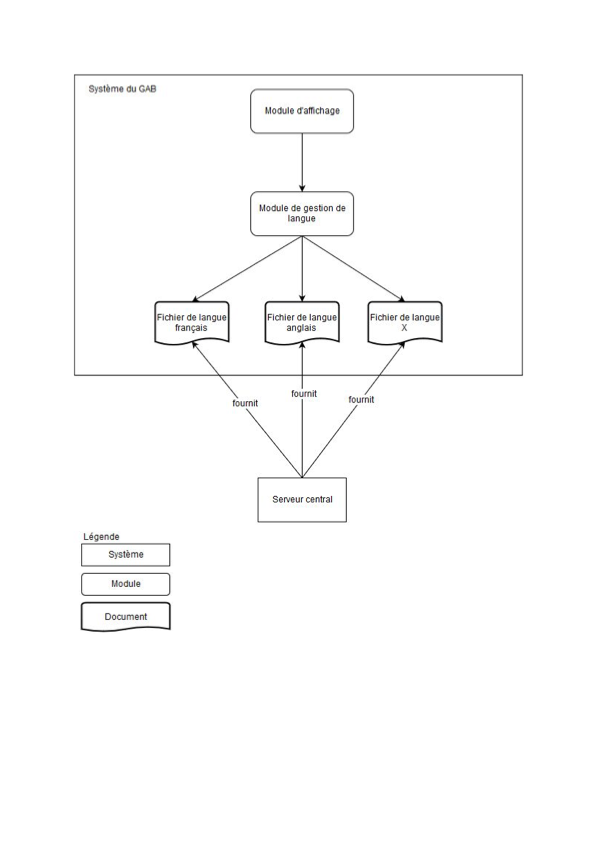
**Réponse**: La langue du système est changée selon le paramètre de langue sélectionné

**Mesure de la réponse**: Avec un module de gestion de langue, l’ajout d’une langue au système est très simple et ne requiert peu ou pas de code

**<Modificabilité> – Tactique 1 (en classe)**

**Description**: Tactique “split module” et augmentation de la cohérence

**Justification**: On sépare le module d’affichage et le module de la langue pour qu’il soit plus facile d’ajouter de nouvelles langues.



Pour faire l’affichage, le module de l’affichage envoie une requête au module de gestion de langue. Celui­ci choisit le bon fichier de ressources à utiliser selon un paramètre de configuration. Les fichiers de langue sont fournis et déployés à distance par le serveur central.

La tactique de “split module” permet de séparer la gestion de la langue du code général de l’affichage.

Groupe 10

**<Modificabilité> – Justification (en classe)**

Le GAB doit être facilement modifiable pour qu’il n’y as pas d’impact sur la disponibilité du GAB pendant une mise à jour du processus de communication.

**<Modificabilité> – Scénario 1 (en classe)**

**Source**: Administrateur du système bancaire.

**Stimuli**: Une mise à jour du processus de communication est nécessaire car le gouvernement fédéral demande d’avoir une connexion supplémentaire à leurs serveurs de surveillance des activités frauduleuses.

**Artéfact**: Code et Source Binaire

**Environnement**: Environnement de maintenance

**Réponse**: Ajouté les modification et les configurations nécessaire pour la communication entre le serveur gouvernemental et le GAB. S’assurer que la communication avec le serveur est sécuritaire et fonctionnelle. Ensuite, faire le déploiement des modifications.

**Mesure de la réponse**: Effort de déploiement

**<Modificabilité> – Tactique 1 (en classe)**

**Description**: Defer bindings

**Justification**: Nous utiliserons une tactique pour différer la configuration actuelle des serveurs à celle des serveurs de détection des activités frauduleuses. De plus, un script est plus rapide et et plus fiable que un être humain. Il n’y aura donc moins ou pas d’impact sur la disponibilité du système.

**<Modificabilité> – Vue Architecturale (en classe)**