Implémentation Idiomes et Meilleures Pratiques

Construire et Maintenir des Applications Clientes Robustes et Évolutives en utilisant PureMVC

À l'aide d'exemples en Actionscript 3 et en MXML

AUTEUR: Cliff Hall <cliffford.hall@puremvc.org>

TRADUIT PAR: Eric La Rocca <eric.la.rocca@puremvc.org>

MISE A JOUR: 4/15/2008

Généralités conceptuelles de PureMVC 4

- Modèle & proxys 4
- Vue & médiateurs 4
- Contrôleur & commandes 5
- Façade & acteurs centraux 5
- Observateurs & notifications 5
- Utiliser des notifications pour déclencher des commandes 6
- Les médiateurs envoient, s'abonnent et reçoivent des notifications 6
- Les proxys envoient mais ne reçoivent pas de notifications 6

Façade 7

- Qu'est-ce qu'une façade concrète? 8
- Créer une façade concrète pour votre application 9
- Initialiser votre façade concrète 11

Notifications 13

- Événements vs. notifications 14
- Définir un événement et des constantes de notification 15

Commandes 16

- Utilisation de commandes macros et de commandes simples 17
- Couplage faible des commandes aux médiateurs et aux proxys 18
- Orchestration d'actions complexes et logique métier 19

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

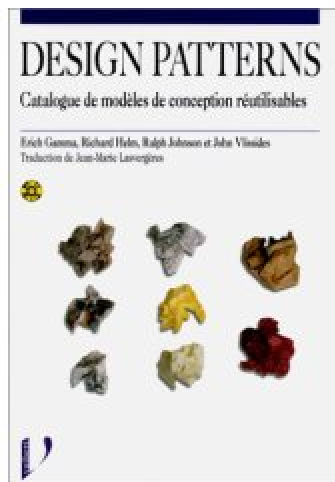
Médiateurs 24

- Responsabilités d'un médiateur concret 24
- Transtyper implicitement un composant visuel 25
- Écouter et répondre au composant visuel 28
- Gérer des notifications dans un médiateur concret 29
- Coupler des médiateurs à des proxys et à d'autres médiateurs 32
- Interactions utilisateur entre composants visuels et médiateurs 34

Proxys 38

- Responsabilités d'un proxy concret 39
- Transtyper implicitement un objet de données 40
- Empêcher le couplage aux médiateurs 42
- Encapsuler la logique domaine dans les proxys 42
- Interagir avec des proxys distants 44

Inspiration



PureMVC est un framework basé sur des patterns et motivé à l'origine par le besoin de concevoir des clients RIA hautement performants. Il fait désormais l'objet de portage vers d'autres langages et plateformes incluant des environnements serveur. Ce document porte sur la partie client.

Alors que l'interprétation et les implémentations sont spécifiques à chaque plateforme supportée par PureMVC, les patterns utilisés sont définis avec précision dans « l'infameux » ouvrage de la « Bande des Quatre » :

Design Patterns: Catalogue des modèles de conception réutilisables (ISBN 2-711-78644-7)

Vivement recommandé.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Généralités conceptuelles de PureMVC

Le framework PureMVC a un objectif précis : vous aider à séparer les parties de code de votre application en trois tiers bien distincts : le modèle, la vue et le contrôleur.

Cette séparation en tiers, ainsi que le degré et le sens de couplage employé pour les faire collaborer est d'une importance capitale pour construire et maintenir des applications évolutives.

Dans cette implémentation du classique méta pattern MVC, ces trois tiers sont gouvernés par trois singletons (classes à instance unique) appelés `Model`, `View` et `Controller`. Ensemble, ces trois classes sont dénommées les «acteurs centraux»

La classe `Facade`, un quatrième singleton, simplifie le développement en fournissant une interface unique pour la communication avec les acteurs centraux.

Modèle & proxys

Le modèle cache les références nommées à des proxys. Le code d'un proxy gère le modèle de données; il communique si besoin est, avec des services distants afin de d'établir un accès persistant ou non avec des données.

Ce qui contribue à faire du modèle un code portable.

Vue & médiateurs

La vue cache les références nommées à des médiateurs. Le médiateur représente les composants visuels; en leur nom, il ajoute des écouteurs, échange des notifications avec le reste du système et manipule directement leur état.

Cela distingue la définition de la vue de la logique qui la contrôle.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Généralités conceptuelles de PureMVC

Contrôleur & commandes

Le contrôleur maintient les 'mappings' nommés aux classes de commande, qui, elles, sont sans état et sont créées au besoin uniquement.

Les commandes peuvent éventuellement accéder ou interagir avec des proxys, envoyer des notifications, exécuter d'autres commandes et sont souvent utilisées pour orchestrer des phases complexes ou d'ampleur comme le démarrage ou la clôture d'une application. Elles abritent la logique métier de votre application.

Façade & acteurs centraux

La façade, qui est un autre singleton, initialise les acteurs centraux (le modèle, la vue, le contrôleur), et fournit un endroit unique pour accéder à toutes leurs méthodes publiques.

En étendant la façade, votre application profite de tous les avantages des acteurs centraux sans avoir à les importer et à les utiliser directement. Vous implémenterez facilement une façade concrète pour votre application, et ce, une seule et unique fois.

Afin d'accéder et de communiquer entre eux, les proxys, les médiateurs et les commandes peuvent alors utiliser la façade concrète de votre application.

Observateurs & notifications

Les applications PureMVC sont prévues pour fonctionner dans des environnements ne disposant pas des classes `Event` et `EventDispatcher` de Flash, le framework implémente donc un mécanisme de notification Observer afin de permettre une communication faiblement couplée entre les acteurs centraux MVC et le reste du système.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Généralités conceptuelles de PureMVC

Observateurs & notifications

Il n'est pas nécessaire de connaître les détails de cette implémentation Observer/Notification; c'est interne au framework. Pour envoyer une notification depuis des proxys, des médiateurs, des commandes et depuis la façade, vous utiliserez une simple méthode qui ne requiert même pas la création d'une instance de la classe `Notification`.

Utiliser des notifications pour déclencher des commandes

Dans votre façade concrète, les commandes sont 'mappées' à des noms de notifications, et elles sont automatiquement exécutées par le contrôleur lorsque les notifications correspondantes sont envoyées. Les commandes orchestrent habituellement des interactions complexes entre des parties de la vue et du modèle tout en n'en sachant le moins possible sur chacune.

Les médiateurs envoient, s'abonnent et reçoivent des notifications

Lorsqu'ils sont enregistrés auprès de la vue, les médiateurs sont interrogés sur les notifications qui les intéressent. A l'appel de leur méthode `listNotifications`, ils doivent retourner un tableau avec les noms des notifications auxquelles ils sont abonnés.

Par la suite, lorsqu'une notification du même nom est envoyée par un acteur quelconque du système, les médiateurs intéressés seront avertis (notifiés) par l'appel de leur méthode `handleNotification` à qui l'on passe une référence de la notification concernée.

Les proxys envoient mais ne reçoivent pas de notifications

Les proxys peuvent envoyer des notifications pour différentes raisons, comme par exemple la réception de la réponse d'un service distant ou bien la mise à jour de données.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Généralités conceptuelles de PureMVC

Les proxys envoient mais ne reçoivent pas de notifications

Pour un proxy, être à l'écoute de notifications implique un couplage trop étroit avec les tiers vue et contrôleur.

Ces tiers étant en charge de représenter le modèle de données et de permettre à l'utilisateur d'interagir avec, ils doivent donc obligatoirement écouter les notifications provenant des proxys qui incarnent ce modèle de données.

Cependant, la vue et le contrôleur devraient pouvoir varier sans affecter le modèle de données.

Par exemple, une application administratrice et une application utilisatrice pourraient partager les classes du même modèle. Si seuls les cas d'utilisation diffèrent, ils peuvent être réalisés via différents couples vue/contrôleur d'un même modèle.

Façade

Les trois acteurs centraux du méta pattern MVC sont représentés en PureMVC par les classes `Model`, `View` et `Controller`. Pour simplifier le processus de développement d'applications, PureMVC emploie le pattern façade.

La façade relaie vos requêtes aux classes `Model`, `View` et `Controller`, afin de vous dispenser d'importer ces classes dans votre code et de devoir les manipuler une à une. La classe `Facade` instancie automatiquement dans son constructeur ces singletons qui composent le cœur MVC.

Habituellement, la façade sera sous-classée dans votre application et utilisée pour initialiser le contrôleur à l'aide d'un 'mapping' de commandes. La préparation du modèle et de la vue est alors orchestrée par les commandes exécutées par le contrôleur.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Façade

Qu'est ce qu'une façade concrète ?

Bien que les acteurs centraux constituent une implémentation complète et utilisable, la façade fournit une implémentation qui devrait être considérée *abstraite,* c'est-à-dire jamais instanciée directement.

Sous-classez plutôt la façade et ajoutez ou supplantez (override) certaines de ses méthodes afin de les rendre utilisables dans votre application.

Cette façade *concrète* est alors utilisée pour accéder et notifier les commandes, les médiateurs et les proxys qui constituent le système. Par convention, on la nomme `ApplicationFacade` mais vous pouvez la nommer comme bon vous semble.

Généralement, la hiérarchie des vues de votre application (composants visuels) sera créée conformément à la plateforme que vous utilisez. En Flex, une application MXML instancie tous ses enfants ou une animation Flash crée tous ses objets sur la scène (Stage). Une fois cette hiérarchie des vues construite, le mécanisme PureMVC est activé et les parties Modèle et Vue sont prêtes à l'emploi.

Par ailleurs, en isolant le code applicatif de l'environnement PureMVC associé, votre façade concrète contribue à faciliter le processus de démarrage. L'application se contente de passer sa propre référence à la méthode 'startup' du singleton qu'est votre Façade concrète.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Façade

Créer une façade concrète pour votre application

Votre Façade concrète n'a pas grand chose à faire pour rendre votre application puissante. Voyez l'implémentation suivante :

ApplicationFacade.as:

```
package com.me.myapp
{
	import org.puremvc.as3.interfaces.*;
	import org.puremvc.as3..patterns.facade.*;

	import com.me.myapp.view.*;
	import com.me.myapp.model.*;
	import com.me.myapp.controller.*;

	// Une façade concrète pour MyApp
	public class ApplicationFacade extends Façade implements IFacade
	{
		// Définition des constantes pour les noms des Notifications
		public static const STARTUP:String		= "startup";
		public static const LOGIN:String		= "login";

		// Méthode Factory du singleton ApplicationFacade
		public static function getInstance() : ApplicationFacade {
			if ( instance == null ) instance = new ApplicationFacade( );
			return instance as ApplicationFacade;
		}

		// Enregistrer les Commandes auprès du Contrôleur
		override protected function initializeController( ) : void {
			super.initializeController();
			registerCommand( STARTUP,  StartupCommand );
			registerCommand( LOGIN,  LoginCommand );
			registerCommand( LoginProxy.LOGIN_SUCCESS,  GetPrefsCommand );
		}
		// Lance l'environnement PureMVC, passage d'une référence à l'application
		public function startup( app:MyApp ) : void
		{
			sendNotification( STARTUP, app );
		}
	}
}
```

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Façade

Créer une façade concrète pour votre application

Quelques points à noter à propos du code précédent:

- Il étend la classe `Facade` de PureMVC, qui, de son coté implémente l'interface `IFacade`.
- Il ne supplante pas le constructeur. Si c'était le cas, il appellerait avant toute chose le constructeur de la superclasse.
- Il définit une méthode statique `getInstance` qui retourne l'instance du singleton. Au besoin, il la crée puis la cache. La référence de l'instance est conservée dans une propriété de la superclasse (`Facade`) et doit être transtypée au type de la sous-classe avant d'être retournée.
- Il définit des constantes pour les noms des notifications. Dans la mesure où la façade concrète est utilisée par les autres éléments pour communiquer, elle est la place idéale pour définir ces constantes.
- Il initialise le contrôleur avec un ensemble de commandes qui seront exécutées lors de l'envoi des notifications attendues.
- Il fournit une méthode `startup` qui prend un argument (dans ce cas) de type `myApp` qu'il passe, via une notification, à `StartupCommand` (enregistrée avec la notification `STARTUP`)

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Façade

Créer une façade concrète pour votre application

Avec ces seules exigences d'implémentation, votre façade concrète va hériter de tout un ensemble de fonctionnalités de sa superclasse abstraite.

Initialiser votre façade concrète

Le constructeur de la façade de PureMVC appelle des méthodes protégées afin d'initialiser les instances des classes `Model`, `View` et `Controller` puis les place en cache pour référence.

Par composition, la façade implémente et expose alors les caractéristiques de `Model`, `View et Controller`; agrégeant leurs fonctionnalités et évitant ainsi au développeur toute interaction directe avec les acteurs centraux du framework.

Où et comment la façade intervient-elle dans l'organisation type d'une application ? Voyez le code `Application` Flex suivant :

MyApp.mxml:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="façade.startup(this)">

    <mx:Script>
    <![CDATA[
            // Récupérer l'ApplicationFacade
            import com.me.myapp.ApplicationFacade;
            private var facade:ApplicationFacade = ApplicationFacade.getInstance();
    ]]>
    </mx:Script>

    <!—Ici le reste de la hiérarchie visuelle  -->
</mx:Application>
```

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Façade

Initialiser votre façade concrète

C'est tout ! Plutôt facile.

Construisez la hiérarchie visuelle de base, récupérez l'instance de `ApplicationFacade` et invoquez sa méthode `startup`.

NOTE: Sous AIR, nous aurions utilisé `applicationComplete`, et en Flash il nous aurait fallu instancier `Facade` puis faire l'appel de `startup` sur le frame 1 ou dans un fichier de classe distinct.

Les points clés de cet exemple sont:

- Nous construisons l'interface MXML de façon classique; en commençant par le tag `<mx:Application>`, avec des composants et des conteneurs standards ou personnalisés.

- Un script déclare et initialise une variable privée avec l'instance du singleton `ApplicationFacade`

- L'initialisation de la variable par un appel à la méthode statique `ApplicationFacade.getInstance` signifie qu'au déclenchement du `creationComplete` du bloc `Application`, la façade aura été créée et, avec elle, le modèle, la vue et le contrôleur; par contre, aucun médiateur ni aucun proxy n'est encore créé à ce stade.

- Dans le `creationComplete` du tag `Application`, nous invoquons la méthode `startup`, en passant une référence de l'application.

Notez qu'habituellement les composants visuels n'ont nul besoin de connaitre ni d'interagir avec la façade, mais le bloc principal `Application` est l'exception à la règle.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Façade

Initialiser votre façade concrète

Le bloc principal `Application` (ou l'animation Flash) construit la hiérarchie visuelle, initialise la façade, puis active le mécanisme PureMVC.

Notifications

PureMVC implémente le pattern Observer afin que les acteurs principaux et leurs collaborateurs puissant communiquer de façon faiblement couplée, et sans dépendance de plateforme.

Le langage Actionscript ne fournit pas le modèle d'événements utilisé par Flex et Flash, ceux-là proviennent du package Flash. Ce framework a été porté vers d'autres plateformes telle que C# et J2ME, car il gère ses propres communications internes et non celles fournies par la plateforme Flash.

Plus qu'un simple remplacement des événements (`Events`), les notifications opèrent de façon fondamentalement différente, et travaillent en synergie avec les événements afin de produire des composants visuels qui soient facilement réutilisables et qui, lorsqu'ils sont correctement construits, ignorent même le système PureMVC auquel ils sont couplés.

Événements vs. notifications

Les événements sont dispatchés depuis des objets d'affichage Flash ceux-là proviennent qui implémentent l'interface `IEventDispatcher`. L'événement est propagé ('bubbled up') à travers la hiérarchie d'affichage, permettant à tous les objets ascendants (parents, grands-parents…) la gestion de cet événement.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Notifications

Événements vs. notifications

C'est un *mécanisme de chaîne de responsabilité* par lequel seuls ceux de la lignée parent/enfant ont la possibilité de recevoir ou d'agir sur un événement à moins qu'ils disposent d'une référence vers le dispatcheur et puissent directement lui assigner un écouteur.

Les notifications sont envoyées par la façade et les proxys; écoutées et envoyées par les médiateurs; 'mappées' et envoyées par les commandes. C'est un mécanisme de publication/souscription par lequel plusieurs observateurs peuvent recevoir ou agir sur la même notification.

Chaque notification peut se voir dotée d'un contenu (body) optionnel, qui peut être un quelconque objet Actionscript.

Contrairement aux événements de Flash, créer une notification personnalisée est rarement nécessaire puisque celle-ci dispose par défaut d'un conteneur. Vous pouvez bien sûr créer des notifications personnalisées afin de renforcer le typage des interactions avec elles, mais choisir entre les bénéfices limités de la vérification à la compilation (particulièrement pour les notifications) et la charge d'avoir à maintenir plusieurs classes de notifications est une question de style de programmation.

Les notifications ont aussi un '`type`' optionnel qui peut aider leurs destinataires à les distinguer. Par exemple, dans une application d'édition de documents. Il peut y avoir une instance de `Proxy` pour chaque document ouvert et un médiateur pour le composant visuel utilisé pour éditer le document. Le proxy et le médiateur pourraient partager une clé unique que le proxy passerait en tant que type d'une notification.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Notifications

Événements vs. notifications

Toutes les instances de médiateur enregistrées pour les notifications de ce proxy seront notifiées, mais utiliseront la propriété 'type' pour déterminer à quelle notification elles devront répondre.

Définir des notifications et des constantes d'événements

Nous avons vu que la façade concrète est une bonne place pour définir les constantes de notifications communes. Constituant le mécanisme central de l'interaction avec le système, toutes les notifications collaboreront de fait avec la façade.

Plutôt qu'une façade concrète, une classe `ApplicationConstants` distinctes est parfois utilisée afin de permettre l'utilisation de ces constantes par une autre application.

Une définition centralisée de constantes pour les noms de notifications garantit qu'en cas de faute de frappe dans le nom d'une notification, le compilateur signalera cette erreur alors qu'il resterait silencieux si l'on utilisait une simple chaine de caractères.

Par contre, ne déclarez pas le nom des événements dans la façade concrète.

Définissez statiquement les constantes de noms d'événements dans chacune des classes frontières qui les génèrent, ou dans les classes des événements personnalisés qui sont dispatchées.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Notifications

Définir des notifications et des constantes d'événements

Représentants les limites physiques de l'application, les composants visuels et les objets de données peuvent demeurer réutilisables s'ils communiquent avec leur médiateur ou proxy associés via un dispatching d'événements plutôt que par des appels directs de méthodes ou l'envoi de notifications.

Si un composant visuel ou un objet de données dispatche un événement que le médiateur ou le proxy associé écoute, il est alors probable que seuls ces paires d'éléments (composant visuel/médiateur ou objet de donnés/proxy) soient intéressées par cet événement en particulier.

Les autres communications entre un écouteur et le reste du système PureMVC devrait se faire via des notifications.

Bien que les relations des ces paires de collaborations soient forcément étroites, elles ont un couplage faible avec le reste de l'application; offrant ainsi plus de maitrise lors d'un éventuel réagencement (refactoring) de l'interface utilisateur ou du modèle de données.

Commandes

La façade concrète initialise habituellement le contrôleur avec l'ensemble des 'mappings' (Notification->Commande) nécessaires au démarrage.

Pour chaque 'mapping', le contrôleur s'enregistre en tant qu'observateur de la notification donnée. Un fois notifié, le contrôleur instancie la commande associée. Finalement, le contrôleur appelle la méthode `execute` de cette commande en lui passant la notification.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Commandes

Les commandes n'ont pas d'état; elles sont créées au besoin et doivent normalement disparaitre après leur exécution. Pour cette raison, il est important de ne pas instancier ni de stocker des références de commandes dans des objets persistants.

Utilisation de commandes macros et de commandes simples

Les commandes comme toute classe du framework PureMVC, implémentent une interface appelée `ICommand`. PureMVC propose deux implémentations que vous pouvez facilement étendre.

La classe `SimpleCommand` dispose juste d'une méthode `execute` qui accepte une instance de `INotification`. Insérez votre code dans la méthode `execute` et c'est tout.

La classe `MacroCommand` vous permet d'exécuter plusieurs sous-commandes en séquence, chacune étant créée et se voit passée la notification par référence.

`MacroCommand` appelle sa méthode `initializeMacroCommand` depuis son constructeur. Pour chaque commande à ajouter, il vous suffit de supplanter (override) cette méthode dans vos sous-classes pour appeler la méthode `addSubCommand`. Vous pouvez créer n'importe quelle combinaison de commandes simples ou de commandes macros.

Couplage faible des commandes aux médiateurs et aux proxys

L'exécution des commandes par le contrôleur est un résultat de l'envoi des notifications. Les commandes ne devraient être instanciées et exécutées que par un contrôleur.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Commandes

Couplage faible des commandes aux médiateurs et aux proxys

Afin de communiquer et d'interagir avec le reste du système, les commandes peuvent:

- enregistrer, supprimer ou vérifier l'enregistrement de médiateurs, de proxys et de commandes.
- Envoyer des notifications destinées à d'autres commandes ou à d'autres médiateurs.
- Récupérer les proxys et les médiateurs et les manipuler directement.

Les commandes nous permettent d'effectuer les changements d'état des éléments visuels, ou d'assurer le transport des données de part et d'autre de la vue.

Elles peuvent être utilisées pour faire des transactions avec le modèle, ce qui engendre de multiples proxys et nécessite l'envoi de notifications lorsque la transaction est terminée, ou pour gérer des exceptions et traiter les incidents.

Orchestration d'actions complexes et logique métier

Avec la multiplication des endroits où placer le code (commandes, médiateurs et proxys); les questions suivantes reviennent inévitablement :

Où placer tel code? Que doit faire une commande exactement ?

La première distinction à faire sur la logique de votre application est entre la logique métier et la logique domaine

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Commandes

Orchestration d'actions complexes et logique métier

Les commandes abritent la logique métier de notre application; l'implémentation technique des cas d'utilisation devrait composer le modèle de domaine. Cela suppose une coordination entre le modèle et la vue.

Le modèle maintient son intégrité à travers l'utilisation de proxys, qui eux abritent la *logique domaine* et exposent une API permettant la manipulation des objets de données. Ces proxys encapsulent tous les accès au modèle de données que cela soit côté client ou côté serveur, ne laissant de pertinent à l'application que ce qui concerne la façon synchrone ou non d'accéder aux données.

Les commandes peuvent être utilisées pour orchestrer des actions complexes qui doivent être exécutées selon un ordre spécifique, et dans certains cas, lorsque le résultat d'une action doit alimenter la suivante.

Les médiateurs et les proxys devraient exposer aux commandes (et entre eux) une interface rudimentaire, qui masque l'implémentation des éléments visuels ou des objets de données auxquels ils sont associés.

Notez que lorsque nous parlons d'un composant visuel, nous parlons d'un bouton ou d'un composant avec lequel l'utilisateur interagit directement. Lorsque nous parlons d'objet de données cela inclut les diverses structures contenant des données ou bien les services distants que nous pouvons appeler pour obtenir ou stocker ces données. *Les commandes interagissent avec les médiateurs et les proxys, mais doivent être isolées des implémentations frontières*. Étudiez les commandes ci-dessous utilisées pour préparer le système :

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Commandes

Orchestration d'actions complexes et logique métier

StartupCommand.as:

```
package com.me.myapp.controller
{
    import org.puremvc.as3.interfaces.*;
    import org.puremvc.as3.patterns.command.*;
    import com.me.myapp.controller.*;

    // une commande macro exécutée lorsque l'application démarre.
    public class StartupCommand extends MacroCommand
    {
        // initialise la macro commande avec des sous-commandes.
        override protected function initializeMacroCommand() : void
        {
            addSubCommand( ModelPrepCommand );

            addSubCommand( ViewPrepCommand );
        }
    }
}
```

C'est une macro commande qui ajoute deux sous commandes, lesquelles sont utilisées selon un ordre de type FIFO (première entrée, première sortie) lorsque `MacroCommand` est exécuté.

Cela constitue une file d'actions à exécuter au démarrage. Mais que devrions-nous faire exactement, et dans quel ordre ?

Avant que l'utilisateur puisse interagir avec les données de l'application, le modèle doit être placé dans un état cohérent et connu. Ensuite, la vue peut être préparée afin de présenter ces données et permettre aux utilisateurs de les manipuler.

Par conséquent, le démarrage consiste habituellement en deux étapes – la préparation du modèle, suivi par la préparation de la vue

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Commandes

Orchestration d'actions complexes et logique métier

ModelPrepCommand.as:

```
package com.me.myapp.controller
{
    import org.puremvc.as3.interfaces.*;
    import org.puremvc.as3.patterns.observer.*;
    import org.puremvc.as3.patterns.command.*;
    import com.me.myapp.*;
    import com.me.myapp.model.*;
    // Créer et enregistrer les Proxys avec le Modèle.
    public class ModelPrepCommand extends SimpleCommand
    {
        // Appelé par  MacroCommand
        override public function execute( note : INotification ) : void
        {
            facade.registerProxy( new SearchProxy() );
            facade.registerProxy( new PrefsProxy() );
            facade.registerProxy( new UsersProxy() );
        }
    }
}
```

Préparer le modèle se résume simplement à créer et à enregistrer tous les proxys requis par le système au démarrage.

`ModelPrepCommand` ci-dessus est une `SimpleCommand` qui prépare le modèle. C'est la première sous-commande de la commande macro (`startupCommand`), elle est donc exécutée en premier.

À travers la façade concrète, on crée et enregistre les différentes classes Proxy que le système utilisera au démarrage. Notez que la commande ne réalise aucune manipulation ou initialisation du modèle de données. Le proxy est responsable de la récupération, de la création ou de l'initialisation des données nécessaires pour la préparation de son objet de donnée associé.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Commandes

Orchestration d'actions complexes et logique métier

ViewPrepCommand.as:

```
package com.me.myapp.controller
{
    import org.puremvc.as3.interfaces.*;
    import org.puremvc.as3.patterns.observer.*;
    import org.puremvc.as3.patterns.command.*;
    import com.me.myapp.*;
    import com.me.myapp.view.*;
    // Crée et enregistre les médiateurs avec la vue.
    public class ViewPrepCommand extends SimpleCommand
    {
        override public function execute( note : INotification ) : void
        {
            var app:MyApp = note.getBody() as MyApp;
            facade.registerMediator( new ApplicationMediator( app ) );
        }
    }
}
```

C'est une commande simple (`SimpleCommand`) qui prépare la vue. C'est la dernière des sous-commandes de la macro commande (`MacroCommand`) et comme telle, elle est exécutée en dernier.

Remarquez que le seul médiateur crée et enregistré est `ApplicationMediator`, qui régit le composant visuel `Application`.

Ensuite, il passe le corps de la notification au constructeur du médiateur. Il s'agit d'une référence à l'objet `Application` transmis par `Application` lui-même lorsque la notification STARTUP a été envoyée (voir l'exemple `myApp` précédent)

`Application` est un composant visuel particulier dans la mesure où il instancie et a pour enfants tous les autres composants visuels qui sont initialisés au démarrage.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Commandes

Orchestration d'actions complexes et logique métier

Pour communiquer avec le reste du système, les composants visuels ont besoin de médiateurs. Et créer ces médiateurs nécessite une référence vers le composant visuel qu'ils vont représenter, et que seul le bloc Application connaît à ce stade.

Le médiateur de `Application` est la seule classe qui sait tout de l'implémentation de `Application`, ceci afin de créer, au sein même de son constructeur, tous les autres médiateurs.

En fait, avec les trois commandes ci-dessus, nous avons orchestrée une initialisation ordonnée du modèle et de la vue. Ainsi, les commandes n'ont savent pas trop sur le modèle et la vue.

Lorsque des éléments du modèle ou de l'implémentation de la vue changent, les proxys et les médiateurs peuvent être ré-agencés à volonté.

Un réagencement des frontières de l'application ne devrait pas impacter la logique métier des commandes.

Le modèle devrait encapsuler la 'logique du domaine', assurant ainsi l'intégrité des données dans les proxys.

Les commandes constituent le 'transactionnel' ou la logique 'affaire' du modèle, encapsulant la coordination des transactions multi-proxys ou en gérant et en rapportant les exceptions d'une façon conforme à ce qu'attend l'application.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Médiateurs

Un médiateur est une classe utilisée pour mettre en relation les interactions de l'utilisateur sur un ou plusieurs composants visuels de l'application (par exemple, une grille Flex ou une animation Flash) avec le reste de l'application PureMVC.

Dans une application Flash, un médiateur place habituellement des écouteurs d'événements dans son composant visuel afin de gérer les actions de l'utilisateur et les requêtes de ce composant. Il envoie et reçoit des notifications pour communiquer avec le reste de l'application.

Responsabilités d'un médiateur concret

Les environnements Flash, Flex et AIR fournissent une vaste gamme de composants visuels très interactifs. Vous pouvez les étendre ou écrire le vôtre en Actionscript afin de présenter le modèle de données de multiples façons et de permettre aux utilisateurs d'interagir avec.

Dans un futur pas si lointain, d'autres plateformes seront capables d'exécuter de l'Actionscript. Et le framework a été porté et présenté sur d'autres plateformes comme Silverlight ou J2ME, élargissant les horizons du développement de RIA avec cette technologie.

Un objectif du framework PureMVC est d'être Indépendant des technologies utilisées aux frontières de l'application et de fournir un langage simple pour adapter le composant d'interface utilisateur ou la structure/service du moment.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Médiateurs

Responsabilités d'un médiateur concret

Pour une application PureMVC, un composant visuel représente n'importe quel composant d'interface utilisateur, indépendamment du framework d'origine, et du nombre de sous-composants qu'il contient. Un composant visuel devrait encapsuler autant d'état et d'opérations que possible, n'exposant qu'une simple API d'événements, de méthodes et de propriétés.

Un médiateur concret nous aide a adapter un ou plusieurs composants visuels à l'application en détenant les seules références à ces composants et en interagissant avec l'API qu'ils exposent.

Les responsabilités du médiateur sont principalement la gestion des événements envoyés par le composant visuel et des notifications pertinentes émises par le reste du système.

Dans la mesure où les médiateurs aussi interagissent souvent avec les proxys, il est fréquent pour un médiateur de récupérer et de conserver dans son constructeur une référence locale des proxys les plus souvent utilisés. Cela réduit les appels répétés à `retrieveProxy` pour obtenir la même référence.

Transtyper implicitement un composant visuel

L'implémentation de base fournie avec PureMVC pour le médiateur accepte un nom et un Objet générique comme seuls arguments du constructeur.

Le constructeur de votre médiateur concret passera son composant visuel à la superclasse qui deviendra immédiatement accessible en tant que propriété protégée nommée `viewComponent` et typée génériquement en `Objet`.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Médiateurs

Transtyper implicitement un composant visuel

Vous pourriez aussi définir dynamiquement le composant visuel du médiateur après sa construction, en appelant sa méthode `setViewComponent`

Quelque soit la manière avec laquelle il a été défini, vous devrez fréquemment transtyper cet `Objet` en son type actuel, afin de pouvoir accéder à son API, ce qui peut devenir lourd et répétitif.

Le langage Actionscript dispose d'accesseurs implicites. Un getter implicite ressemble à une méthode, mais apparait comme une propriété au reste de la classe et de l'application. C'est très utile pour régler les fréquents problèmes de transtypage.

Une recette pratique pour votre médiateur concret est d'utiliser un getter implicite qui transtype un composant visuel en son type actuel et de lui donner un nom significatif.

En créant une méthode comme celle-ci :

```
protected function get controlBar() : MyAppControlBar
{
        return viewComponent as MyAppControlBar;
}
```

Puis, quelque part dans votre médiateur, *plutôt que de faire*:

```
MyAppControlBar ( viewComponent ).searchSelection =
MyAppControlBar.NONE_SELECTED;
```

Nous pouvons plutôt faire cela:

```
controlBar.searchSelection = MyAppControlBar.NONE_SELECTED;
```

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Médiateurs

Écouter et répondre au composant visuel

Un médiateur est habituellement en charge d'un seul composant visuel, mais il pourrait en gérer plusieurs, comme `ApplicationToolBar` et ses boutons ou ses contrôles intégrés.

Nous pouvons placer un ensemble de contrôles (tel un formulaire) dans un composant visuel puis les exposer au médiateur sous la forme de propriétés de ce composant. Mais le mieux serait d'encapsuler autant que possible l'implémentation du composant et d'exposer les propriétés du composant via un objet de type personnalisé.

Le médiateur va gérer les échanges entre le tiers contrôleur et le tiers modèle, actualisant le composant visuel à la réception des notifications attendues.

Dans Flash, à la construction du médiateur ou à l'appel de sa méthode `setViewComponent`, nous plaçons habituellement des écouteurs d'événements dans le composant visuel en utilisant la méthode:

```
controlBar.addEventListener( AppControlBar.BEGIN_SEARCH, onBeginSearch );
```

La nature de la réaction du médiateur en réponse à cet événement, est bien sûr, totalement définie par les exigences du moment.

En réponse à un événement, un médiateur concret réalise généralement certaines des actions suivantes :

- Vérifier, si nécessaire, le type ou le contenu de l'événement.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Médiateurs

Écouter et répondre au composant visuel

- Vérifier ou modifier les propriétés (ou les méthodes) exposées d'un composant visuel.

- Vérifier ou modifier les propriétés (ou les méthodes) exposées d'un proxy.

- Émettre une ou plusieurs notifications qui seront traitées par d'autres médiateurs ou des commandes (voire par la même instance de médiateur émetteur).

Quelques règles empiriques:

- Si d'autres médiateurs doivent être impactés lors de la réponse à un événement, mettez alors à jour un proxy commun ou envoyez une notification qui sera traitée de façon appropriée par chacun de ces médiateurs.

- Si les interactions entre les médiateurs sont nombreuses, une bonne façon est de recourir à une commande afin de coder les différentes étapes a réaliser en un même endroit.

- Évitez de récupérer et de travailler directement sur les autres médiateurs ou de faire un médiateur qui le permette.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Médiateurs

Écouter et répondre au composant visuel

- Pour manipuler et diffuser aux médiateurs les informations concernant l'état de l'application, définissez des valeurs ou appelez des méthodes dans des proxys spécialement créés pour conserver cet état et permettez aux médiateurs d'écouter les notifications émises par ces proxys.

Gérer des notifications dans un médiateur concret

Contrairement à l'ajout d'écouteurs dans les composants visuels, la technique de couplage du médiateur au système PureMVC est simple et automatique.

À l'enregistrement de la vue, le médiateur est interrogé sur l'intérêt qu'il porte aux notifications. Il répond en fournissant un tableau contenant le nom de toutes les notifications auxquelles il est abonné.

La façon la plus simple de répondre est à l'aide d'une expression unique qui créée et retourne un tableau anonyme contenant les noms de notifications. Ces noms devraient être des constantes statiques, habituellement définies dans la façade concrète.

Définir une liste d'abonnement aux notifications d'un médiateur est facile:

```
override public function listNotificationInterests() : Array
{
    return [  ApplicationFacade.SEARCH_FAILED,
              ApplicationFacade.SEARCH_SUCCESS
           ];
}
```

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Médiateurs

Gérer des notifications dans un médiateur concret

Lorsqu'une des notifications listée est émise par un quelconque acteur du système (incluant le médiateur lui-même), la méthode `handleNotification` du médiateur sera appelée et la notification passée en argument.

De par sa lisibilité et la facilité avec laquelle on peut mettre à jour la gestion des notifications dans la méthode `handleNotification`, la structure switch/case est préférable à la structure if/else if.

Essentiellement, la réponse à une notification devrait se limiter à peu de code, et toute l'information requise devrait se trouver dans la notification elle-même. Occasionnellement, certaines données peuvent provenir d'un proxy basé sur l'information fournie par la notification mais globalement la logique de traitement d'une notification doit rester simple. Si ça n'est pas le cas, cela indique que vous essayez de placer la logique métier relevant d'une commande dans le gestionnaire de notification de votre médiateur.

```
override public function handleNotification( note : INotification ) : void
{
    switch ( note.getName() )
    {
            case ApplicationFacade.SEARCH_FAILED:
                    controlBar.status = AppControlBar.STATUS_FAILED;
                    controlBar.searchText.setFocus();
                    break;

            case ApplicationFacade.SEARCH_SUCCESS:
                    controlBar.status = AppControlBar.STATUS_SUCCESS;
                    break;

    }
}
```

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Médiateurs

Gérer des notifications dans un médiateur concret

En outre, le gestionnaire de notification d'un médiateur ne doit pas gérer plus de 4 ou 5 notifications.

Dans le cas contraire, cela indique que les responsabilités du médiateur devraient être divisées plus finement. Créez des médiateurs pour les sous-composants du composant visuel plutôt que de tenter de tous les gérer dans un médiateur monolithique.

L'utilisation d'une méthode de notification unique et prédéfinie constitue la différence fondamentale entre la façon qu'a un médiateur d'écouter des événements et celle qu'il a d'écouter des notifications.

Dans le cas des événements, nous avons un certain nombre de méthodes gestionnaires; habituellement une pour chaque événement que le médiateur doit gérer. Généralement ces méthodes se contentent d'envoyer des notifications et ne devraient ni être complexes, ni gérer dans le détail le composant visuel. Ce dernier devrait encapsuler les détails d'implémentation, en exposant une API sommaire au médiateur.

Dans le cas des notifications, nous avons un gestionnaire unique dans lequel nous gérons toutes les notifications qui intéressent le médiateur.

Le mieux est de placer dans la méthode `handleNotification` *la totalité du code répondant aux notifications. Chaque notification étant séparément traitée à l'aide d'un* `case` *dans une structure* `switch`*.*

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Médiateurs

Gérer des notifications dans un médiateur concret

Il y a eu beaucoup de discussion autour de l'usage d'un 'switch/case' dans la mesure où de nombreux développeurs considèrent cette approche comme limitée puisque tous ces 'case' s'exécutent dans la même méthode et ont donc la même portée. Néanmoins, cette approche a été spécifiquement choisie afin de limiter l'usage d'un médiateur; elle demeure donc l'approche recommandée.

Le médiateur est destiné à permettre la communication entre l'élément visuel et le reste du système.

Considérons le rôle d'un interprète assurant une conversation entre son ambassadeur et le reste des membres lors d'une conférence à l'ONU. Ses taches devraient se limiter à faire des traductions, à faire suivre des messages et occasionnellement à trouver une métaphore appropriée ou une information particulière. C'est la même chose pour le rôle du médiateur dans PureMVC.

Coupler des médiateurs à des proxys et à d'autres médiateurs

Dans la mesure où la vue est ultimement chargée de représenter le modèle de données de façon graphique et interactive, nous pouvons supposer un couplage unidirectionnel et relativement fort avec les proxys de l'application. La vue doit connaître le modèle, mais le modèle n'a rien à savoir de la vue.

Les médiateurs peuvent librement accéder aux proxys du modèle, lire et manipuler les objets de données via une quelconque API exposée par le proxy. Cependant, déplacer ce traitement dans une commande réduira le couplage entre la vue et le modèle.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Médiateurs

Coupler des médiateurs à des proxys et à d'autres médiateurs

De la même manière, les médiateurs pourraient récupérer depuis la vue, des références à d'autres médiateurs, les lire et les manipuler de toutes les façons exposées par le médiateur récupéré.

Néanmoins cela n'est pas une pratique recommandée dans la mesure où cela crée des dépendances entre différentes parties de la vue, nuisant ainsi à la capacité de ré-agencer une partie de la vue sans en affecter une autre.

Un médiateur qui souhaite communiquer avec une autre partie de la vue devrait envoyer une notification plutôt que de récupérer et de manipuler directement un autre médiateur.

Les médiateurs ne devraient pas exposer des méthodes permettant à des tiers de manipuler leur(s) composant(s) visuel(s); ils devraient plutôt le faire eux-mêmes en réponse à des notifications.

Si l'essentiel des manipulations internes d'un composant visuel est effectué dans un médiateur (en réponse à un événement ou à une notification), ré-agencez (refactor) ce traitement en une méthode interne au composant, cela aura pour effet d'encapsuler autant que possible ses implémentations et d'augmenter ainsi sa capacité à être réutilisé.

Si l'essentiel des manipulations des proxys ou de leurs données est effectué dans un médiateur, ré-agencez le tout en une commande, cela aura pour effet d'alléger le médiateur, de déplacer la logique métier vers des commandes alors réutilisables par d'autres parties de la vue, et finalement cela diminuera le couplage entre la vue et le modèle.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Médiateurs

Interactions utilisateur entre composants visuels et médiateurs

Considérons un composant `LoginPanel` comportant un formulaire. Nous avons un `LoginPanelMediator` qui reçoit de l'utilisateur via le `LoginPanel`, une identification (identifiant + mot de passe) et une demande de login et qui, en réponse, initie une demande de connexion.

La collaboration entre le composant `LoginPanel` et le `LoginPanelMediator` consiste en l'envoi par le composant d'un événement `TRY_LOGIN` lorsque l'utilisateur s'identifie et souhaite se connecter. Le `LoginPanelMediator` réagit à l'événement par l'envoi d'une notification avec pour contenu le 'value objet' `LoginVO` complété par le composant.

LoginPanel.mxml:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml"
    title="Login" status="{loginStatus}">
    <!—Les événements que ce composant diffuse. Nous ne pouvons malheureusement
    pas utiliser de constantes ici, car Metadata est une directive du compilateur -->
    <mx:MetaData>
            [Event('tryLogin')];
    </mx:MetaData>
    <mx:Script>
    <![CDATA[
            import com.me.myapp.model.vo.LoginVO;
            // Les champs de ce formulaire sont bi directionnellement
            //reliés aux propriétés de l'objet
            [Bindable] public var loginVO:LoginVO = new LoginVO();
            [Bindable] public var loginStatus:String = NOT_LOGGED_IN;
            //Définir une constante dans le composant visuel pour les //noms
            d'événements
            public static const TRY_LOGIN:String='tryLogin';
            public static const LOGGED_IN:String='Logged In';
            public static const NOT_LOGGED_IN:String='Enter Credentials';
    ]]>
    </mx:Script>
```

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

```
    <mx:Binding source="username.text" destination="loginVO.username"/>
    <mx:Binding source="password.text" destination="loginVO.password"/>

    <!—Le  formulaire de Login-->
    <mx:Form id="loginForm" >
        <mx:FormItem label="Username:">
                <mx:TextInput id="username" text="{loginVO.username}" />
        </mx:FormItem>
        <mx:FormItem label="Password:">
         <mx:TextInput id="password" text="{loginVO.password}"
           displayAsPassword="true" />
        </mx:FormItem>
        <mx:FormItem >
           <mx:Button label="Login" enabled="{loginStatus == NOT_LOGGED_IN}"
                click="dispatchEvent( new Event(TRY_LOGIN, true ));"/>
        </mx:FormItem>
   </mx:Form>
</mx:Panel>
```

Le composant visuel `LoginPanel` complète un nouveau `LoginVO` avec les entrées du formulaire puis diffuse un événement lorsque le bouton 'Login' est cliqué. `LoginPanelMediator` le récupère alors.

Cela cantonne le composant visuel au simple rôle de collecteur de données alertant le système lorsque celles-ci sont prêtes.

Un composant plus abouti pourrait, par exemple, activer le bouton login uniquement lorsque l'identifiant et le mot de passe ont tout deux été saisis, empêchant ainsi toute tentative incomplète de connexion.

Le composant visuel masque son implémentation interne, la totalité de son API utilisée par le médiateur se résumant en un événement `TRY_LOGIN`, une propriété de `LoginVO` à vérifier et la propriété `status` du composant `Panel`

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Médiateurs

Interactions utilisateur entre composants visuels et médiateurs

Le `LoginPanelMediator` va aussi répondre aux notifications `LOGIN_FAILED` et `LOGIN_SUCCESS` puis va ajuster le statut de `LoginPanel`.

LoginPanelMediator.as:

```
package com.me.myapp.view
{
    import flash.events.Event;
    import org.puremvc.as3.interfaces.*;
    import org.puremvc.as3.patterns.mediator.Mediator;
    import com.me.myapp.model.LoginProxy;
    import com.me.myapp.model.vo.LoginVO;
    import com.me.myapp.ApplicationFacade;
    import com.me.myapp.view.components.LoginPanel;

    // Un médiateur pour interagir avec le composant LoginPanel.
    public class LoginPanelMediator extends Mediator implements IMediator
    {

            public static const NAME:String = 'LoginPanelMediator';
            public function LoginPanelMediator( viewComponent:LoginPanel )
            {
                    super( NAME, viewComponent );
                    LoginPanel.addEventListener( LoginPanel.TRY_LOGIN, onTryLogin );
            }
            // Liste les notifications attendues
            override public function listNotificationInterests( ) : Array {
                    return [    LoginProxy.LOGIN_FAILED,
                                LoginProxy.LOGIN_SUCCESS    ];
            }

            // Gère les notifications
            override public function handleNotification( note:INotification ):void
            {
                    switch ( note.getName() )  {
                            case LoginProxy.LOGIN_FAILED:
                                    LoginPanel.loginVO = new LoginVO( );
                                    loginPanel.loginStatus = LoginPanel.NOT_LOGGED_IN;
                                    break;
```

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Médiateurs

Interactions utilisateur entre composants visuels et médiateurs

```
                    case LoginProxy.LOGIN_SUCCESS:
                        loginPanel.loginStatus = LoginPanel.LOGGED_IN;
                        break;
                }
            }
            // L utlisateur a cliqué le bouton Login: tentative de login
            private function onTryLogin ( event:Event ) : void  {
                sendNotification( ApplicationFacade.LOGIN, loginPanel.loginVO );
            }
            // Transtypage du viewComponent en son type actuel
            protected function get loginPanel() : LoginPanel  {
                return viewComponent as LoginPanel;
            }
    }
}
```

Notez que `LoginPanelMediator` place un écouteur d'événement dans le constructeur de `LoginPanel` afin que la méthode `onTryLogin` soit invoquée lorsque l'utilisateur clique sur le bouton Login.

Dans la méthode `onTryLogin`, la notification `LOGIN` est nvoyée, et avec elle, `LoginVO` qui contient les informations concernant l'utilisateur.

Plus tôt dans la façade de l'application, nous avions associé la commande `LoginCommand` à la notification `LOGIN` à l'aide d'une commande `registerCommand`.

La méthode `execute` de cette commande va alors invoquer la méthode `login` de `LoginProxy` en passant l'objet `LoginVO`. `LoginProxy` tentera alors une connexion via un service distant puis enverra une notification `LOGIN_SUCCESS` ou `LOGIN_FAILED`. Ces classes sont définies plus loin à la fin de la section sur les proxys.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Médiateurs

Interactions utilisateur entre composants visuels et médiateurs

La méthode `listNotificationInterests` indique que `loginPanelMediator` est abonné à deux notifications: `LOGIN_SUCCESS` et `LOGIN_FAILED`, Une fois notifié, il fixera la variable `loginStatus` de `LoginPanel` à `LOGGED_IN` en cas de succès ou effacera l'objet `LoginVO` et fixera `loginStatus` à `NOT_LOGGED_IN` dans le cas d'un échec.

Proxys

En général, le pattern proxy consiste à fournir un substitut à un objet afin de contrôler l'accès à celui-ci. Dans une application basée sur PureMVC, la classe `Proxy` est utilisée spécifiquement pour gérer une partie du modèle de données de l'application.

Habituellement, un proxy accède à une structure de données créée localement et d'une complexité arbitraire. C'est l'objet de donnée (DO) du `Proxy`.

Dans ce cas, les moyens pour interagir avec cet objet impliquent probablement des accesseurs synchrones aux données. On peut exposer tout ou partie des propriétés et méthodes de l'objet de données ou une référence à celui-ci. En plus d'exposer des méthodes pour actualiser les données, il est aussi possible d'envoyer au reste du système, des notifications signalant la modification de ces données.

Un proxy distant pourrait être utilisé pour encapsuler l'interaction avec un service distant en charge de sauvegarder ou de récupérer un bloc de données. Le proxy peut conserver l'objet communiquant avec le service distant et contrôler l'accès aux données échangées avec ce service.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Proxys

Dans un tel cas, on doit pouvoir définir une donnée ou appeler une méthode du proxy puis attendre une notification asynchrone émise par le proxy lorsque le service reçoit la donnée.

Responsabilités d'un proxy concret

Le proxy concret nous permet d'encapsuler une partie du modèle de données, d'où qu'elle vienne et quel que soit son type, en gérant un objet de données ainsi que son accès par l'application.

L'implémentation de `Proxy` fournie avec PureMVC est un simple objet conteneur de données qui peut être enregistré avec le modèle.

Bien qu'elle soit parfaitement utilisable sous cette forme, vous sous-classerez plutôt la classe `Proxy` et ajouterez ensuite les fonctionnalités spécifiques à un proxy particulier.

Les variations habituelles du pattern Proxy sont :

- *Proxy distant, dans lequel les données gérées par le proxy concret se trouvent à distance et sont manipulés via un service quelconque.*

- *Proxy et délégué, ou l'accès à un objet service doit être partagé entre différents proxys. La classe Delegate maintient l'objet service, contrôle son accès et s'assure ainsi que les réponses sont acheminées aux bons requérants.*

- *Proxy de protection, utilisé lorsque les objets doivent avoir différents droits d'accès.*

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Proxys

Responsabilités d'un proxy concret

- *Proxy virtuel, qui crée à la demande des objets volumineux ou complexes.*

- *Proxy intelligent, qui charge l'objet de données en mémoire lors de l'accès initial, gère un compteur de références et permet son verrouillage afin de prévenir toute modification par un autre objet.*

Transtyper implicitement un objet de données

L'implémentation de base du Proxy fournie avec PureMVC accepte comme arguments du constructeur un nom et un objet générique Vous pouvez dynamiquement définir l'objet de données du proxy après sa construction via la méthode `setData`.

Comme pour le médiateur et son composant visuel, vous devrez fréquemment transtyper cet `Objet` en son type actuel, afin de pouvoir accéder aux méthodes et propriétés qu'il expose; c'est lourd et répétitif et risque de mener à des constructions exposant inutilement l'implémentation de l'objet de données.

Aussi, comme l'objet de données est une structure souvent complexe, nous avons besoin de références nommées pour les différentes parties de la structure ainsi qu'une référence pointant sur la structure elle-même.

A nouveau, les accesseurs implicites d'Actionscript se révèlent très pratiques pour éviter les fréquents transtypage et les problèmes insoupçonnés d'implémentation.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Proxys

Transtyper implicitement un objet de données

Dans votre proxy concret, utilisez un getter implicite qui transtype l'objet de données en son type courant et donnez-lui un nom significatif.

Par ailleurs, il est possible de définir plusieurs getters de types différents pour récupérer des parties spécifiques de l'objet de données.

Par exemple:

```
public function get searchResultAC () : ArrayCollection
{
        return data as ArrayCollection;
}

public function getResultEntry( index:int ) : SearchResultVO
{
        return searchResultAC.getItemAt( index )  as SearchResultVO;
}
```

Dans le médiateur, plutôt que:

```
var item:SearchResultVO =
ArrayCollection ( searchProxy.getData() ).lastResult.getItemAt( 1 ) as SearchResultVO;
```

Nous pouvons écrire:

```
var item:SearchResultVO = searchProxy.resultEntry( 1 );
```

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Proxys

Empêcher le couplage aux médiateurs

Contrairement au médiateur, le proxy n'étant pas concerné par l'état de la vue, il n'est jamais interrogé sur les notifications qui l'intéresse et n'est jamais averti (notifié) de l'envoi d'une d'entre-elles. Le proxy expose plutôt des méthodes et des propriétés destinées à être utilisées par les autres acteurs.

Le proxy concret ne devrait pas utiliser les médiateurs pour informer le système des changements de l'objet de données.

À la place, il devrait envoyer des notifications qui seront traitées par les commandes ou les médiateurs. La façon dont le système est affecté par ces notifications ne devrait pas avoir de conséquence sur le proxy.

En gardant le modèle indépendant des implémentations du système, la vue et le contrôleur peuvent être ré-agencés sans affecter le modèle.

L'inverse n'est pas totalement vrai. Le modèle peut difficilement changer sans affecter la vue, ni le contrôleur éventuellement. Après tout, ces tiers n'existent que pour permettre à l'utilisateur d'interagir avec le modèle.

Encapsuler la logique domaine dans les proxys

Un changement dans le modèle impliquera souvent un réagencement des parties vue/contrôleur. *En nous assurant de placer dans les proxys le maximum de la logique domaine, nous augmentons le degré de séparation entre le modèle d'une part et les intérêts combinés de la vue et du contrôleur d'autre part.*

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Proxys

Encapsuler la logique domaine dans les proxys

Le proxy peut être utilisé non seulement pour contrôler l'accès aux données mais aussi pour les traiter afin de les garder dans un état valide.

Par exemple, le calcul d'une taxe de vente est une fonction de la logique domaine et devrait donc se trouver dans un proxy, et non dans un médiateur ni dans une commande.

Bien que cette fonction puisse se trouver dans n'importe lequel de ces tiers, le choix du proxy n'est pas seulement logique mais permet un réagencement plus léger et plus facile des autres tiers.

Un médiateur peut récupérer le proxy; appeler sa fonction de calcul de taxe, lui passer éventuellement les éléments d'un formulaire. Mais placer ce traitement dans le médiateur équivaudrait à placer la logique domaine dans la vue. Ce calcul est une règle du modèle de domaine et la vue ne le voit que comme une propriété du modèle de domaine, disponible lorsque les valeurs d'entrées sont fournies.

Imaginez que vous travaillez sur une RIA prévue pour fonctionner dans le navigateur internet d'un ordinateur de bureau. Une nouvelle version, pour PDA cette fois, doit être développée avec un jeu de fonctionnalités réduit, mais avec l'intégralité des exigences du modèle de l'application actuelle.

Avec une séparation adéquate des responsabilités, nous devrions pouvoir réutiliser le modèle dans sa totalité et lui adjoindre uniquement une nouvelle vue et un nouveau contrôleur.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Proxys

Encapsuler la logique domaine dans les proxys

Placer la fonction de calcul dans le médiateur peut sembler efficace ou facile lors de l'implémentation ; vous avez simplement pris les données depuis un formulaire pour les traiter et les placer dans le modèle.

Toutefois, pour chaque version de votre application, vous devrez alors dupliquer vos efforts ou copier/coller la logique de calcul de la taxe de vente dans une nouvelle vue complètement différente, plutôt que d'y accéder automatiquement par l'inclusion d'une librairie constituant votre modèle.

Interagir avec des proxys distants

Un proxy distant est simplement un proxy qui obtient son objet de données depuis une source distante. Cela signifie habituellement que nous interagissons avec cet objet de façon asynchrone.

La façon utilisée pour que le proxy obtienne ses données dépend de la plateforme client, de l'implémentation du service distant, et des préférences du développeur. Dans un environnement Flash/Flex, nous pourrions employer `HTTPService`, `WebService`, `RemoteObject`, `DataService` ou encore `XMLSocket` afin de lancer des requêtes de service depuis un proxy.

Selon les spécifications, un proxy distant peut soit transmettre des requête dynamiquement, en réponse à la définition d'une propriété ou à l'appel d'une méthode; soit faire une requête unique lors de sa construction puis fournir ensuite un accès aux données récupérées.

Plusieurs optimisations peuvent être appliquées au proxy pour accroître l'efficacité de la communication avec un service distant.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Proxys

Interagir avec des proxys distants

On peut le construire en mettant en cache les données afin de réduire les « bavardages » réseau; ou en envoyant des mises à jour aux seuls blocs de données ayant été modifiés, et ce afin de réduire la consommation de la bande passante.

Si une requête est dynamiquement invoquée sur un proxy distant par un autre acteur du système, le proxy doit pouvoir alors envoyer une notification à la réception des données.

Les parties intéressées par cette notification ne sont pas forcément celles qui ont initiées la requête.

Par exemple, l'invocation d'une recherche via un service distant et l'affichage du résultat pourrait se faire selon les étapes suivantes :

- Un composant visuel initie une recherche en diffusant un événement.

- Son médiateur répond en récupérant le proxy distant approprié et en définissant la propriété `searchCriteria`

- La propriété `searchCriteria` du proxy est en fait un setter implicite qui stocke la valeur, initialise la recherche via un objet `HTTPService` puis écoute cet objet en attente d'un événement signalant un résultat ou un incident.

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Proxys

Interagir avec des proxys distants

- Le service diffuse en retour un événement `ResultEvent` auquel le proxy répond en plaçant l'objet résultant dans une de ses propriétés publiques (data).

- Le proxy envoie alors une notification indiquant le succès du service et place dans son contenu une référence à l'objet de données.

- Un autre médiateur a précédemment manifesté son intérêt pour cette notification et il y répond en assignant les données obtenues à une propriété `dataProvider` du composant visuel associé.

Ou considérez un LoginProxy qui détient un 'value objet' LoginVO (une classe ne contenant que des données). Ce loginVO pourrait ressembler à cela :

```
package com.me.myapp.model.vo
{
        // Mappe le VO à la classe distante suivante
        [RemoteClass(alias="com.me.myapp.model.vo.LoginVO")]

        [Bindable]
        public class LoginVO
        {
            public var username: String;
            public var password: String;
            public var authToken: String; // défini par le serveur si l'identification est valide
        }
}
```

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Proxys

Interagir avec des proxys distants

LoginProxy expose des méthodes pour définir l'identification, gérer les sessions (login/logout) et récupérer les jetons d'autorisation inclus dans les appels au service qui suivent le login conformément à la logique d'authentification suivante :

LoginProxy:

```
package com.me.myapp.model
{
    import mx.rpc.events.FaultEvent;
    import mx.rpc.events.ResultEvent;
    import mx.rpc.remoting.RemoteObject;
    import org.puremvc.as3.interfaces.*;
    import org.puremvc.as3.patterns.proxy.Proxy;
    import com.me.myapp.model.vo.LoginVO;

    // un proxy pour logger l'utilisateur
    public class LoginProxy extends Proxy implements IProxy {
        public static const NAME:String             = 'LoginProxy';
        public static const LOGIN_SUCCESS:String    = 'loginSuccess';
        public static const LOGIN_FAILED:String     = 'loginFailed';
        public static const LOGGED_OUT:String       = 'loggedOut';
        private var loginService: RemoteObject;

        public function LoginProxy ()  {
            super( NAME, new LoginVO ( ) );
            loginService = new RemoteObject();
            loginService.source = "LoginService";
            loginService.destination = "GenericDestination";
            loginService.addEventListener( FaultEvent.FAULT, onFault );
            loginService.login.addEventListener( ResultEvent.RESULT, onResult );
        }

        // Transtypage de l'objet de données avec un getter implicite
        public function get loginVO( ) : LoginVO {
            return data as LoginVO;
        }
```

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Proxys

Interagir avec des proxys distants

```
            // L'utilisateur est loggé si le VO login contient un jeton d'autorisation
            public function get loggedIn():Boolean {
                    return ( authToken != null );
            }

            // les appels aux services subséquents  doivent inclure un jeton d'autorisation
            public function get authToken():String {
                    return loginVO.authToken;
            }

            //définit l'identification, lance le login ou le logout puis  réessaie
            public function login( tryLogin:LoginVO ) : void {
                    if ( ! loggedIn )    {
                            loginVO.username= tryLogin.username;
                            loginVO.password = tryLogin.password;
                    } else {
                            logout();
                            login( tryLogin );
                    }
            }

            //pour se délogger, réinitialiser LoginVO
            public function logout( ) : void {
                    if ( loggedIn ) loginVO = new LoginVO( );
                    sendNotification( LOGGED_OUT );
            }

            //notifie un succès de login
            private function onResult( event:ResultEvent ) : void {
                    setData( event.result ); // immédiatement disponible en tant que loginVO
                    sendNotification( LOGIN_SUCCESS, authToken );
            }

            //notifie un échec de login
            private function onFault( event:FaultEvent) : void  {
                    sendNotification( LOGIN_FAILED,  event.fault.faultString );
            }
        }
}
```

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Proxys

Interagir avec des proxys distants

Un LoginCommand pourrait récupérer le LoginProxy, définir l'identification et invoquer la méthode login, appelant le service.

Un GetPrefsCommand pourrait répondre à la notification LOGIN_SUCCESS, récupérer le authToken contenu dans la notification et faire un appel au prochain service qui récupère les préférences de l'utilisateur.

LoginCommand:

```
package com.me.myapp.controller
{
    import org.puremvc.as3.interfaces.*;
    import org.puremvc.as3.patterns.command.*;
    import com.me.myapp.model.LoginProxy;
    import com.me.myapp.model.vo.LoginVO;

    public class LoginCommand extends SimpleCommand
    {
        override public function execute( note: INotification ) : void
        {
            var loginVO : LoginVO = note.getBody() as LoginVO;
            var loginProxy: LoginProxy;
            loginProxy = facade.retrieveProxy( LoginProxy.NAME ) as LoginProxy;
            loginProxy.login( loginVO );
        }
    }
}
```

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.

Proxys

Interagir avec des proxys distants

GetPrefsCommand:

```
package com.me.myapp.controller
{
    import org.puremvc.as3.interfaces.*;
    import org.puremvc.as3.patterns.command.*;
    import com.me.myapp.model.LoginProxy;
    import com.me.myapp.model.vo.LoginVO;

    public class GetPrefsCommand extends SimpleCommand
    {
            override public function execute( note: INotification ) : void
            {
                    var authToken : String = note.getBody() as String;
                    var prefsProxy : PrefsProxy;
                    prefsProxy = facade.retrieveProxy( PrefsProxy.NAME ) as PrefsProxy;
                    prefsProxy.getPrefs( authToken );
            }
    }
}
```

PureMVC est un framework open-source et gratuit créé et géré par Futurescale, Inc. Copyright © 2006-08, Droits réservés. La réutilisation est gérée par le Creative Commons 3.0 Attribution US License. PureMVC, incluant le présent document, tout support de formation ou code source de démonstration téléchargés depuis le site de Futurescale sont fournis en l'état sans garantie d'aucune sorte, expresse ou implicite, incluant notamment les garanties implicites d'adéquation, ou la garantie de non violation.