ubiflow

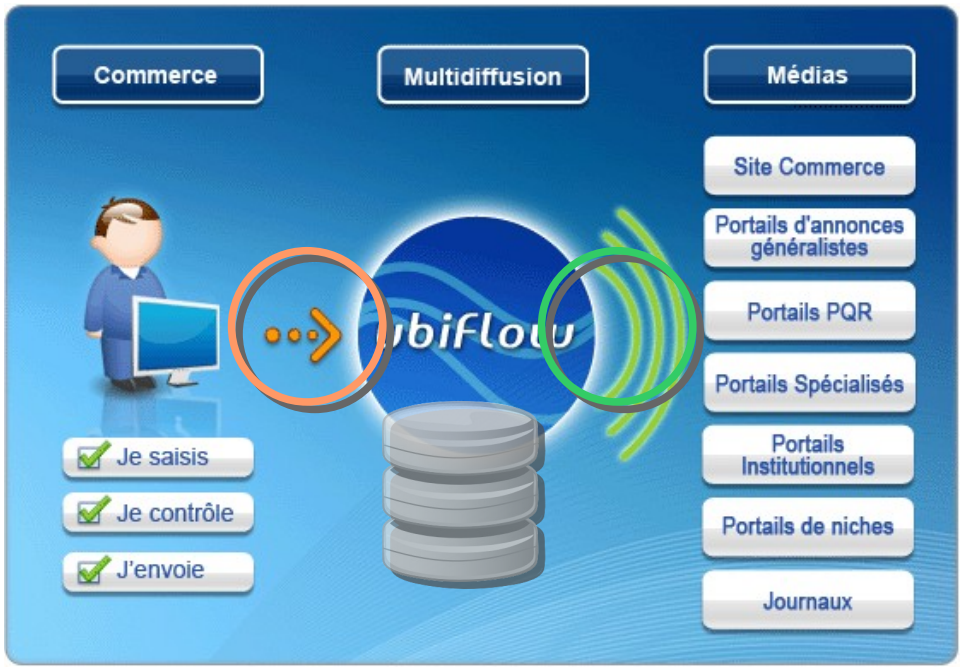
« Architecture GWT (Google Web Toolkit) : bonnes pratiques pour applications professionnelles »

grafotech
Les forums de granit

Luc Sorel – Dr. ingénieur R&D Ubiflow
luc.sorel@ubiflow.net

Présentation

Ubiflow *Apporteur d'Audience*



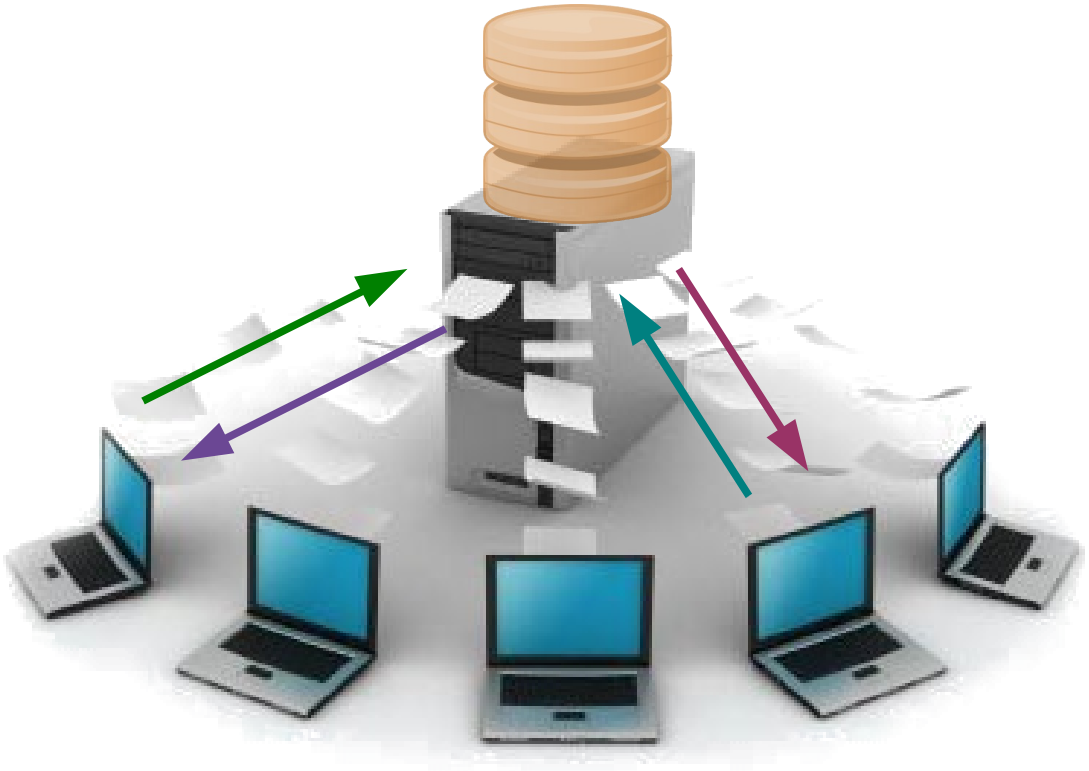
Multidiffusion d'annonces

articles dans logiciel métier → annonces sur des médias

Backoffices

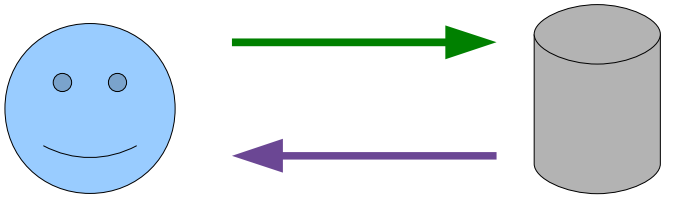
annonceur : sélection annonces / médias
partenaire : suivi activité des annonceurs

Contexte application web



données gérées sur un serveur

mode **requête** **réponse** HTTP

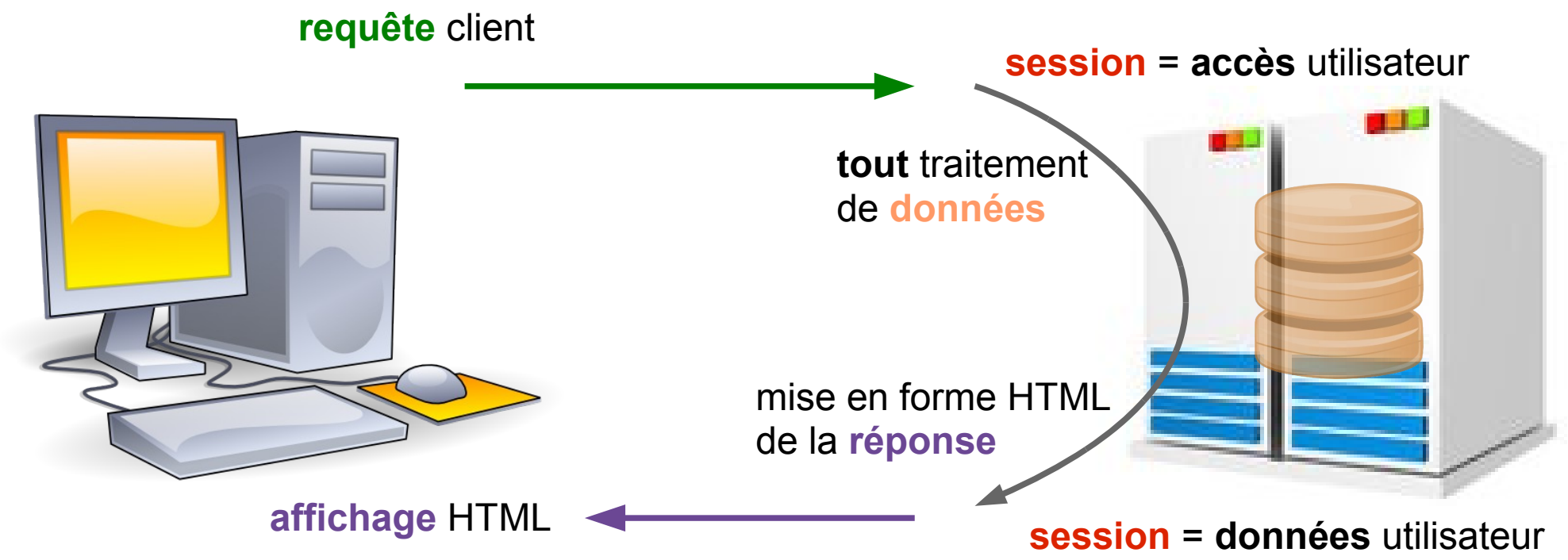


sessions utilisateur :

- données « partagées »
- état personnalisé de l'application

Contexte application web

Approche « serveur d'application »

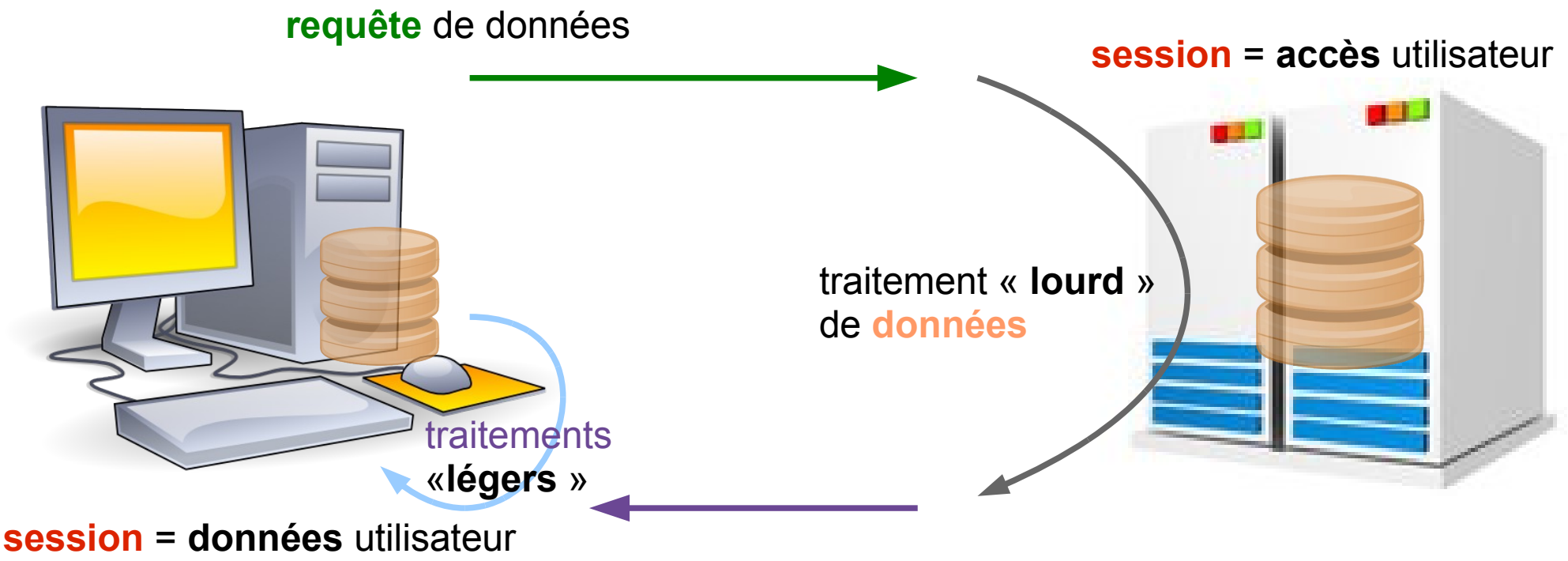


Navigateur : statique, en pause pendant la requête

Serveur : charge de travail élevée, état dynamique

Contexte application web

Approche « application internet riche »



Navigateur : «applicatif », dynamique

Serveur : charge de travail réduite, état « statique »

Google Web Toolkit ?

Homogénéiser le développement d'applications Javascript

Projet Java

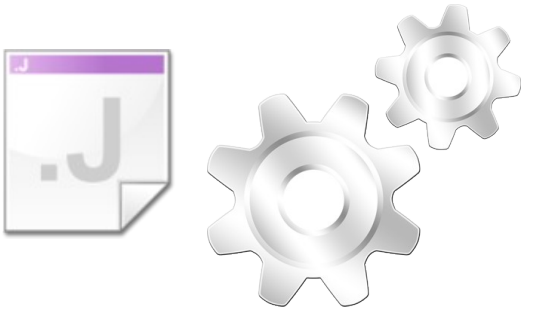
JRE « réduit »
+
bibliothèques GWT dédiées

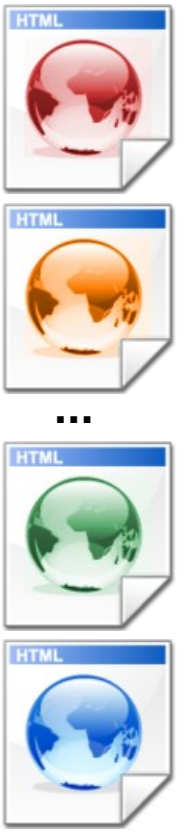
Compilateur GWT

Application Javascript

permutations « navigateurs »

fichier « pilote »

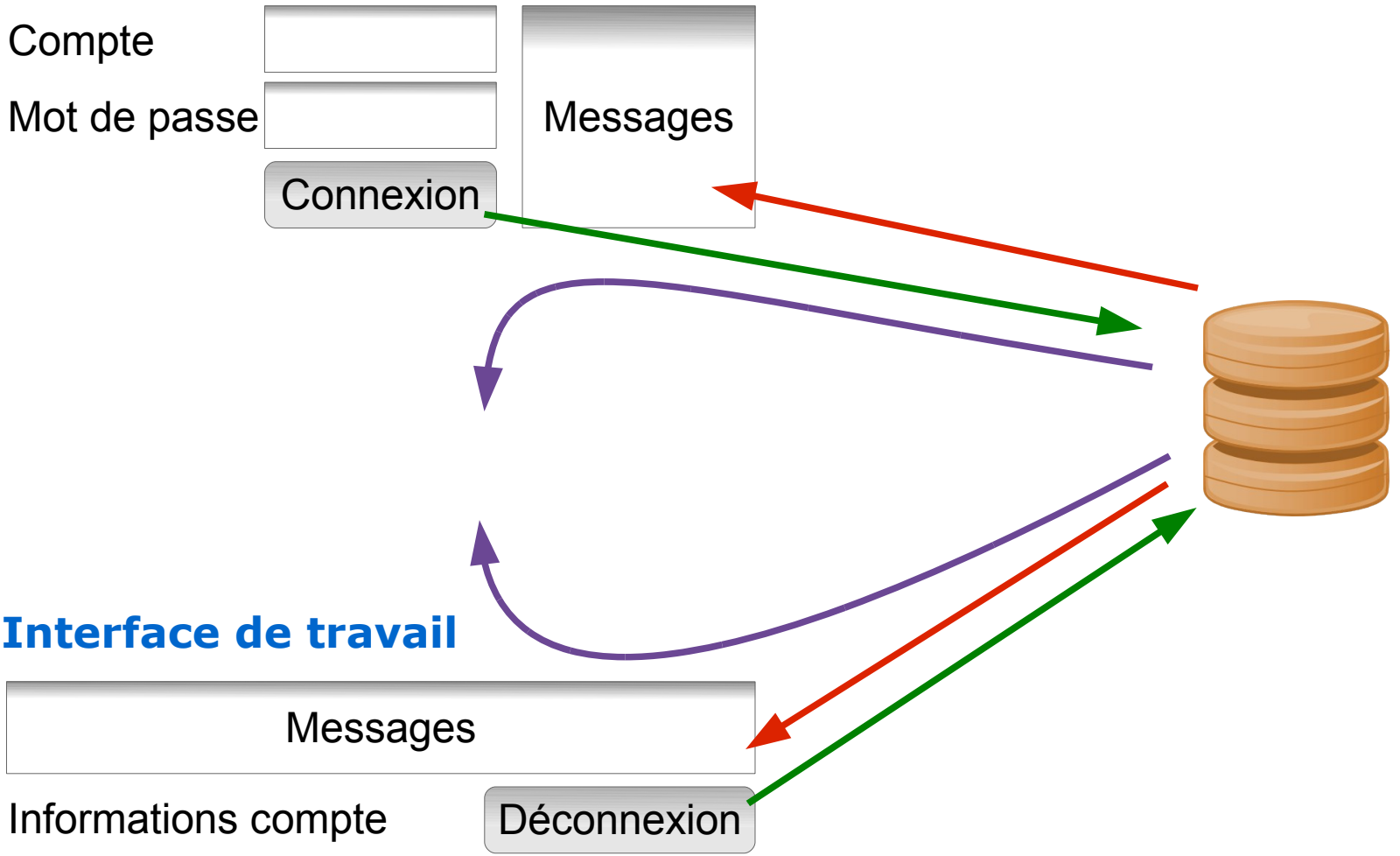




Démo 1

Application test : **Hespéris**

Interface de connexion



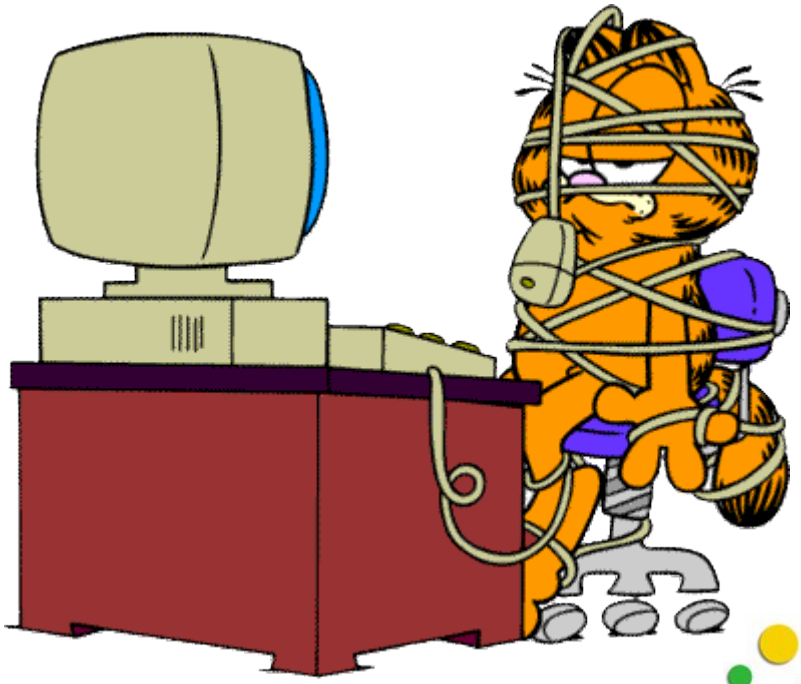
Bilan Démo 1

Ça marche mais...

Les classes *Connexion* et *Deconnexion* doivent « connaitre » l'application (*réutilisation ?*)

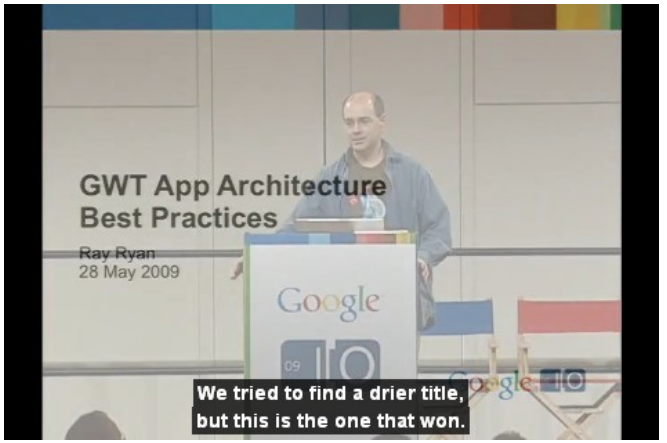
Beaucoup de classes internes : morcellement de contexte (*maintenance ?*)

Échanges de données avec le serveur pilotés par les widgets (*contrôle ?*)



Google IO convention 2009

Conférence de **Ray Ryan** : *Best Practices For Architecting Your GWT App*



Points clefs :

- suivre une architecture Modèle – Vue – Présenteur
- utiliser un bus d'événements
- utiliser l'injection de dépendances

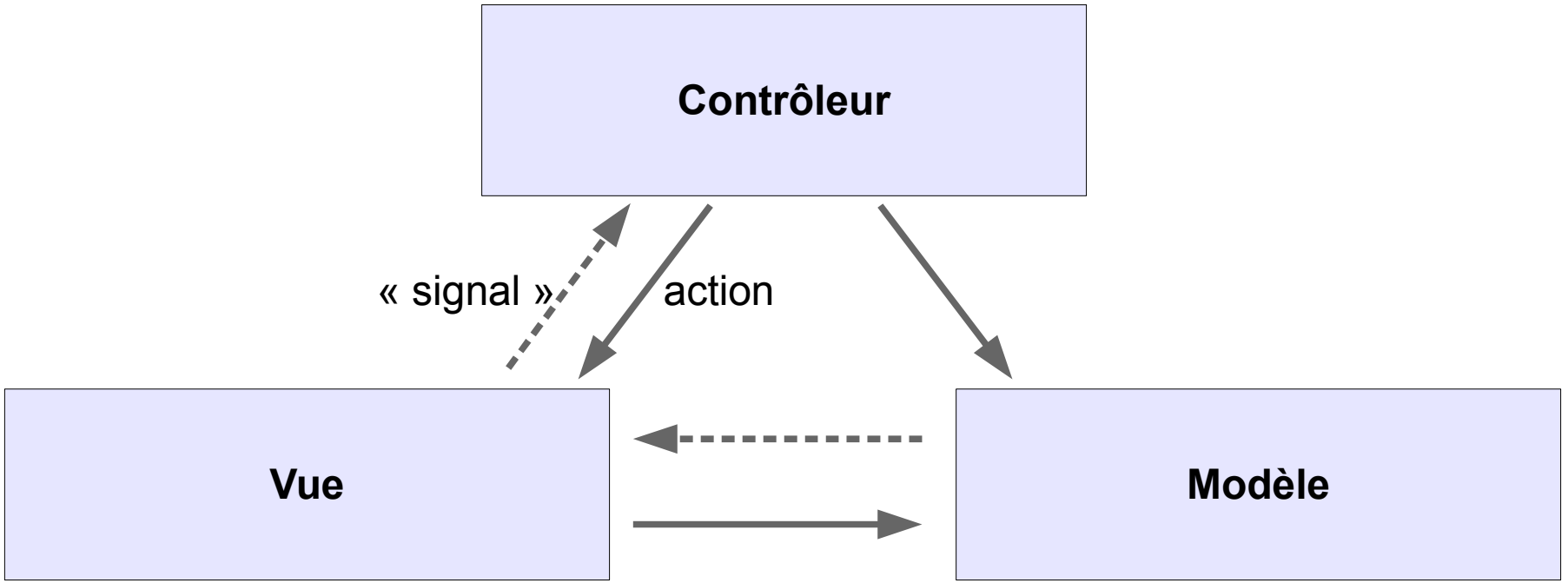
- si souhaité, utiliser le gestionnaire d'historique dès la conception de l'application
- communication avec le serveur : utiliser le design pattern Command (RPC par ex.)
- MVP et tests fonctionnels

ubiflow

Parlons de nos patrons...

Patron de conception MVP

Patron de conception classique **Modèle – Vue - Contrôleur**



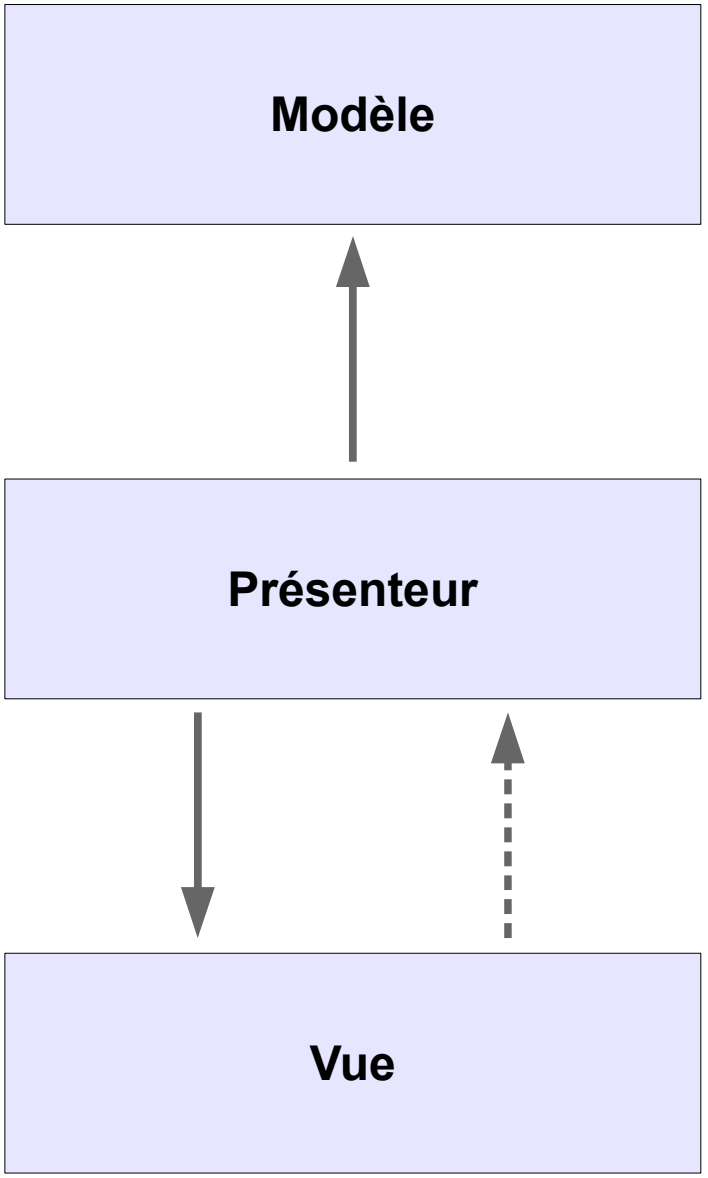
L'approche Modèle permet le développement de tests unitaires

Maintenance des cohérences entre Vue et Contrôleur

Tests fonctionnels sur le Contrôleur ET la Vue

Patron de conception MVP

Patron de conception **Modèle – Vue - Présenteur**



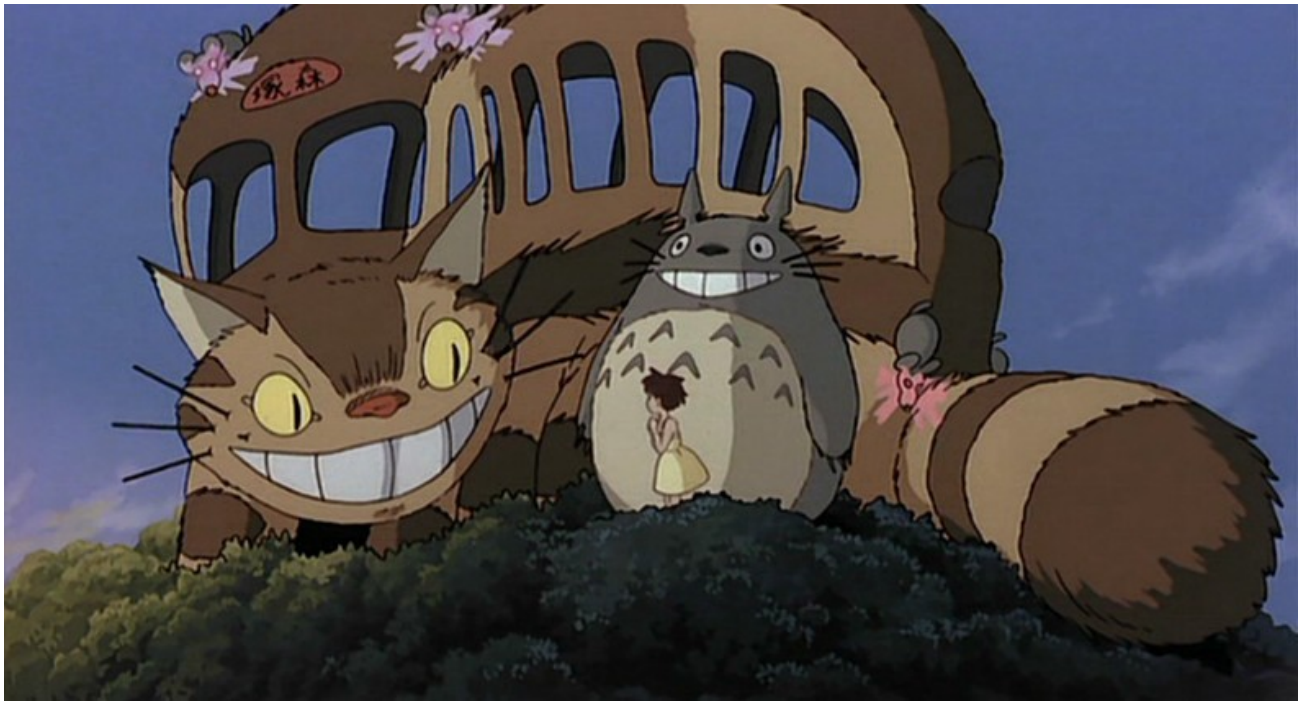
Le **Modèle** se concentre sur
la gestion des données

Le **Vue** se concentre sur
l'affichage de valeurs et
la capture d'actions de l'utilisateur

Les tests fonctionnels ne concernent
plus que le **Présenteur**

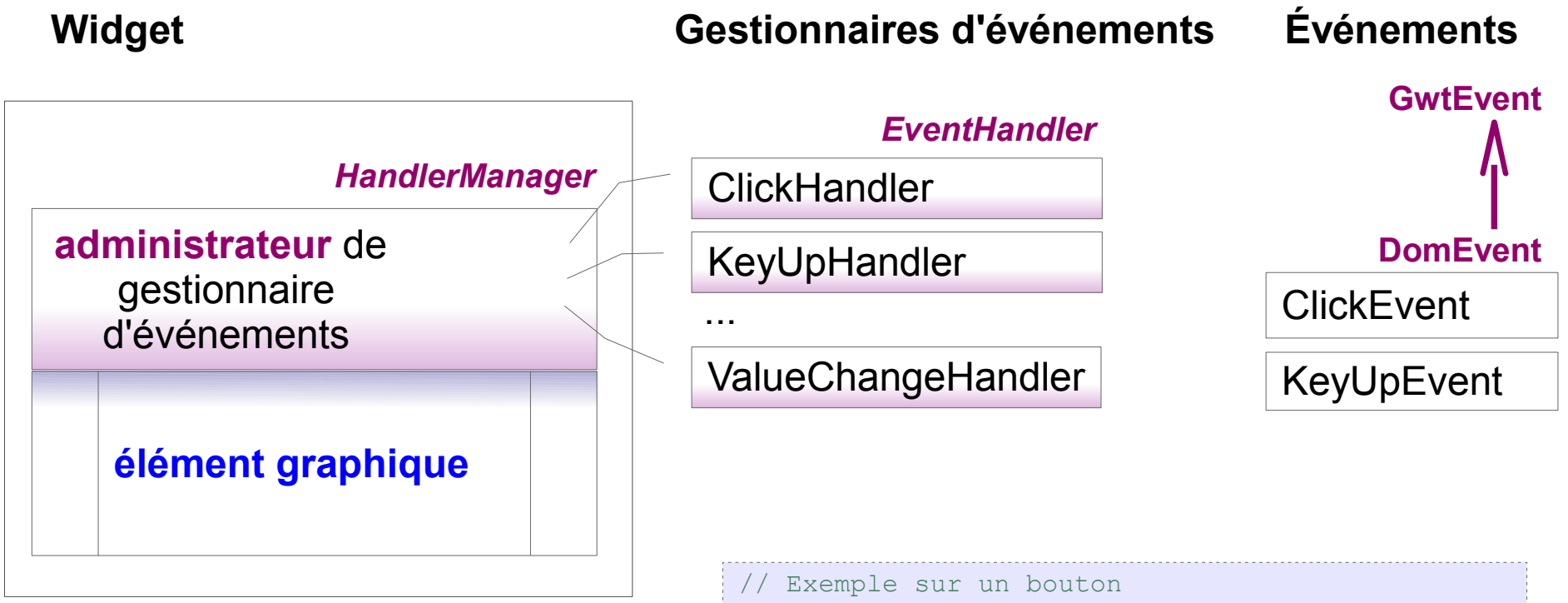
ubiflow

Prenons le bus !
(tous ensemble...)



Bus d'événements

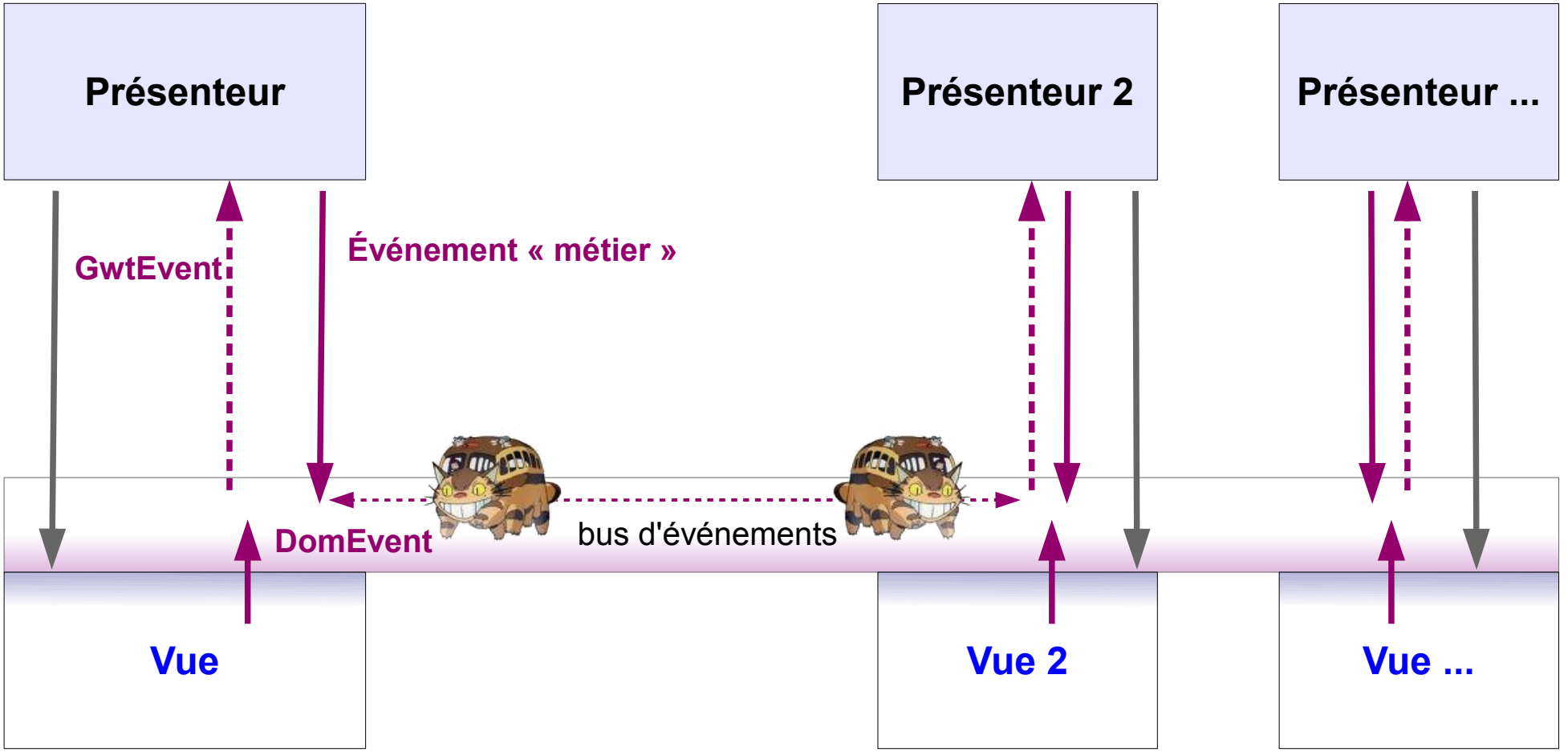
Rappels : concepts événementiels des widgets




```
// Exemple sur un bouton
iB_connexion.addClickHandler(new ClickHandler()
{
      @Override
      public void onClick(ClickEvent event)
      {
            // TODO Auto-generated method stub
      }
});
```

Bus d'événements

Objectif : partager l'administrateur de gestionnaires d'événements



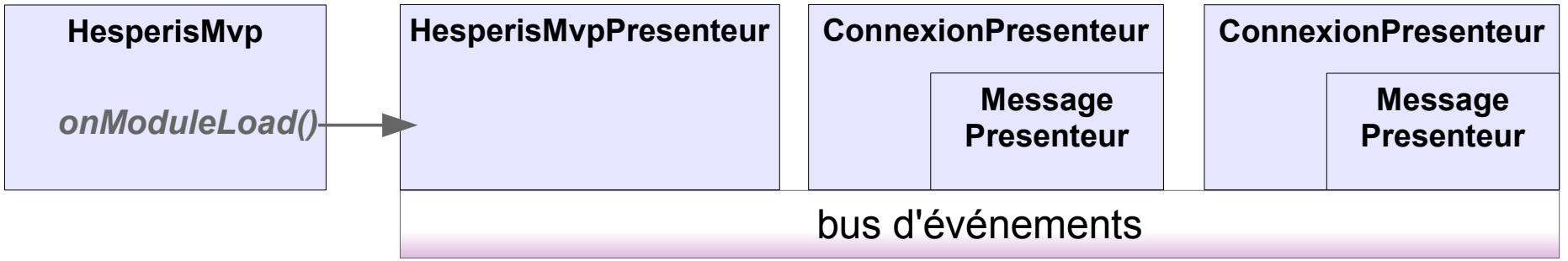
Conversion des événements « DOM » en **événements « métier »**

Communication entre **présenteurs** via les événements « métier »

Démo 2

Refonte : Hespéris sauce MVP

Utilisation de la bibliothèque **gwt-mvp** développée par Eduardo S. Nunes



Démo 2

Refonte : Hespéris sauce MVP

Vue :

- composition des widgets élémentaires (labels, boutons, champs de formulaire, etc.)
- aucun gestionnaire d'événement !
- **initWidget(...) :** définition du widget englobant qui « contient » la vue
- **asWidget() :** méthode d'accès à la vue
- méthodes de renvoi des widgets élémentaires à travers leurs interfaces événementielles

Démo 2

Refonte : Hespéris sauce MVP

Présenteur :

- construction : **bus d'événements** commun, **vue** associée
- **bind(...)**
  - ajoute les gestionnaires d'événements à la vue → événements « métier »
  - définit les événements « métier » écoutés et comment les gérer
- méthodes *internes* de traitement des événements
- méthodes *exposées* aux autres présenteurs
- **unbind(…)** : à la destruction supprime les gestionnaires d'événements et ceux des présenteurs incorporés

Événement « métier »

Classe de définition de l'événement *« demande de connexion »*

```
public class DemandeConnexionEvent extends GwtEvent<DemandeConnexionHandler>
{
	private static Type<DemandeConnexionHandler> TYPE;
	private String is_login;
	private String is_motDePasse;
	// Constructeur de DemandeConnexionEvent
	public DemandeConnexionEvent( String ps_login, String ps_motDePasse )
	{
		is_login = ps_login;
		is_motDePasse = ps_motDePasse;
	}

	// Accesseurs
	public String getLogin()
	{ return is_login; }
	public String getMotDePasse()
	{ return is_motDePasse; }

	// Méthodes requises pour l'association avec le gestionnaire d'événements
	public static Type<DemandeConnexionHandler> getType()
	{
		return (TYPE != null) ? TYPE : (TYPE = new Type<DemandeConnexionHandler>());
	}
	@Override
	public final Type<DemandeConnexionHandler> getAssociatedType()
	{
		return getType();
	}
	@Override
	protected void dispatch( DemandeConnexionHandler pH_handler )
	{
		pH_handler.onDemandeConnexion( this );
	}
}
```

Événement « métier »

Transformation d'un clic en événement « métier »

```
public class ConnexionPresenter extends BasePresenter<Display> implements IConnexion
{
	//...
	public void bind()
	{
		//...
		// Capte l'événement DOM de clic sur le bouton [Connexion]
		registerHandler( display.getConnexionClickHandlers().addClickHandler(
				new ClickHandler()
				{
					public void onClick( ClickEvent pE_clickEvent )
					{
						envoyerDemandeConnexionEvent();
					}
				}));
	}

	// Envoie l'événement métier DemandeConnexionEvent
	private void envoyerDemandeConnexionEvent()
	{
		eventBus.fireEvent(
				new DemandeConnexionEvent(
					display.getLoginTexte().getText(),
					display.getMotDePasseTexte().getText() )
		);
	}
}
```

Événement « métier »

Classe de définition du gestionnaire d'événement « métier »

```
// Déclaration de l'interface du handler gérant les événements DemandeConnexionEvent
public interface DemandeConnexionHandler extends EventHandler
{
	/*
	 * Déclaration de la méthode gérant les événements DemandeConnexionEvent
	 */
	void onDemandeConnexion( DemandeConnexionEvent pE_demandeConnexion );
}
```

Traitement de l'événement « métier »

```
public class HesperisMvpPresenter extends BasePresenter<Display> implements IHesperisMvp
{
	//...
	public void bind()
	{
		//...
		// Capte les événements de type DemandeConnexionEvent pour lancer la méthode de traitement
		registerHandler( eventBus.addHandler(DemandeConnexionEvent.getType(),
				new DemandeConnexionHandler() {
					@Override
					public void onDemandeConnexion(DemandeConnexionEvent pE_DemandeConnexion)
					{
						doDemanderConnexion( pE_DemandeConnexion );
					}
				}) );
	}

	//Demande une connexion d'utilisateur au serveur
	protected void doDemanderConnexion( DemandeConnexionEvent pE_demandeConnexion )
	{
		//À suivre
	}
}
```

Événement « métier »

Traitement de l'événement « métier »

```
public class HesperisMvpPresenter extends BasePresenter<Display> implements IHesperisMvp
{
	//...
	public void bind()
	{
		registerHandler( eventBus.addHandler(DemandeConnexionEvent.getType(),
				new DemandeConnexionHandler() {
					@Override
					public void onDemandeConnexion(DemandeConnexionEvent pE_DemandeConnexion)
					{
						doDemanderConnexion( pE_DemandeConnexion );
					}
				}) );
	}
	//Demande une connexion d'utilisateur au serveur
	protected void doDemanderConnexion( DemandeConnexionEvent pE_demandeConnexion )
	{
		//...
		String ls_demandeConnexion = "controleur.php?"
			+ "action=connecter&login=" + pE_demandeConnexion.getLogin()
			+ "&password=" + pE_demandeConnexion.getMotDePasse();
		ls_demandeConnexion = URL.encode(ls_demandeConnexion);

		// envoi de la requête avec la méthode POST
		RequestBuilder lRB_requeteDemandeConnexion = new RequestBuilder( RequestBuilder.POST,
			ls_demandeConnexion );
		try
		{
			lRB_requeteDemandeConnexion.sendRequest( null,
				ConnexionRequestCallback.Instance(eventBus) );
		}
		catch( RequestException pRE_exception )
		{
			eventBus.fireEvent( new MessageEvent(
					new Erreur("Requête de connexion erronnée :\n"
					+ pRE_exception.getMessage())
			) );
		}
	}
}
```

Injection de dépendances

Injection de dépendances

Bibliothèques GIN (Google INjection)

Définition des dépendances (*HesperisMvpModule.java*)

- bus d'événements commun
- association entre Présenteurs et Vues (interfaces)

Définition de l'injecteur (*HesperisMvpInjector.java*)

- accès au présenteur général
- accès au bus d'événements

Au chargement de l'application (*HesperisMvp.java*)

- création de l'injecteur général utilisé
- fournisseur du Présenteur général

Conclusions

Modèle – Vue – Présenteur :

découplage des classes, simplification, structuration

Bus d'événements :

communication « métier » unifiée entre modules

Injection de dépendances :

instanciation indépendante des modules

Facilitation des tests !

Maintenance du code !

Réutilisation des modules !

Perspectives...

- historique GWT
- framework de test (en mimant les vues)
- patron de conception « Command »
- atomisation des données échangées avec le serveur
- ...

Ressources web

Eclipse PHP Development Tools http://www.eclipse.org/pdt/

Plugin Google pour Eclipse http://code.google.com/intl/fr/eclipse/docs/getting_started.html

JRE Emulation Reference
http://code.google.com/intl/fr/webtoolkit/doc/latest/RefJreEmulation.html

Conférence Google 2009 : Best Practices For Architecting Your GWT App
http://code.google.com/intl/fr/events/io/2009/sessions/GoogleWebToolkitBestPractices.html

Bibliothèque *gwt-mvp* utilisée (et GIN) http://code.google.com/p/gwt-mvp/

Alternative : *gwt-presenter* (non testée) http://code.google.com/p/gwt-presenter/

Présentation et ressources en ligne
http://www.lucsorel.com/index.php?page=downloads#gwt-mvp