Rapport

Du

Projet Météo

Objet du document	Rapport du projet Météo (avec utilisation de composants)
Destinataires du document	Didier Hoareau
Binôme	Jérôme Catric, Clément Le Ny, Emmanuel Meheut, Yitian Yang
Date	05/01/2007

SOMMAIRE

1 PRÉSENTATION DU PROJET..... 3

2 ARCHITECTURE DE L'APPLICATION MÉTÉO 3

2.1 Diagramme de composants.....3
2.2 Description des différents composants.....5
2.2.1 Le composant « Parseur ».....5
2.2.2 Le composant « AeroVille ».....5
2.2.3 Le composant « Serveur ».....5
2.2.4 Le composant « Pressure ».....5
2.2.5 Le composant « Wind ».....6
2.2.6 Le composant « Weather ».....6
2.2.7 Le composant « Temperature ».....6
2.2.8 Le composant « Visibility ».....6
2.2.9 Le composant « Prevision ».....6
2.2.10 Le composant « Client ».....7
2.2.11 Le composant « Affichage ».....7

3 RÉFLEXION SUR L'APPROCHE PAR COMPOSANTS..... 8

3.1 Limites de la programmation par objets..... 8
3.2 la programmation par composants 8
3.2.1 Les inconvénients de la programmation par composants.....8
3.2.2 Les avantages de la programmation par composants.....9

1 Présentation du projet

Ce projet a pour objectif de réaliser l'application station météo, dans le cadre de l'unité d'enseignement INF2104.

Le logiciel météo est une application qui fournit des informations météorologiques sur une région choisie au préalable. Pour cela, elle est à l'écoute de bornes météo, situées dans les aéroports, qui fournissent de l'information météorologique à l'état brut.

Le rôle de cette application est d'exploiter cette information et d'indiquer, pour le lieu choisi, les éléments suivants :

- le temps : Ensoleillé, Peu nuageux, Nuageux, Couvert, Pluvieux ;
- la température : définie en degré Celsius (°C) ;
- la pression atmosphérique : définie en hectopascal (hpa) ;
- la vitesse du vent ;
- l'humidité de l'air.

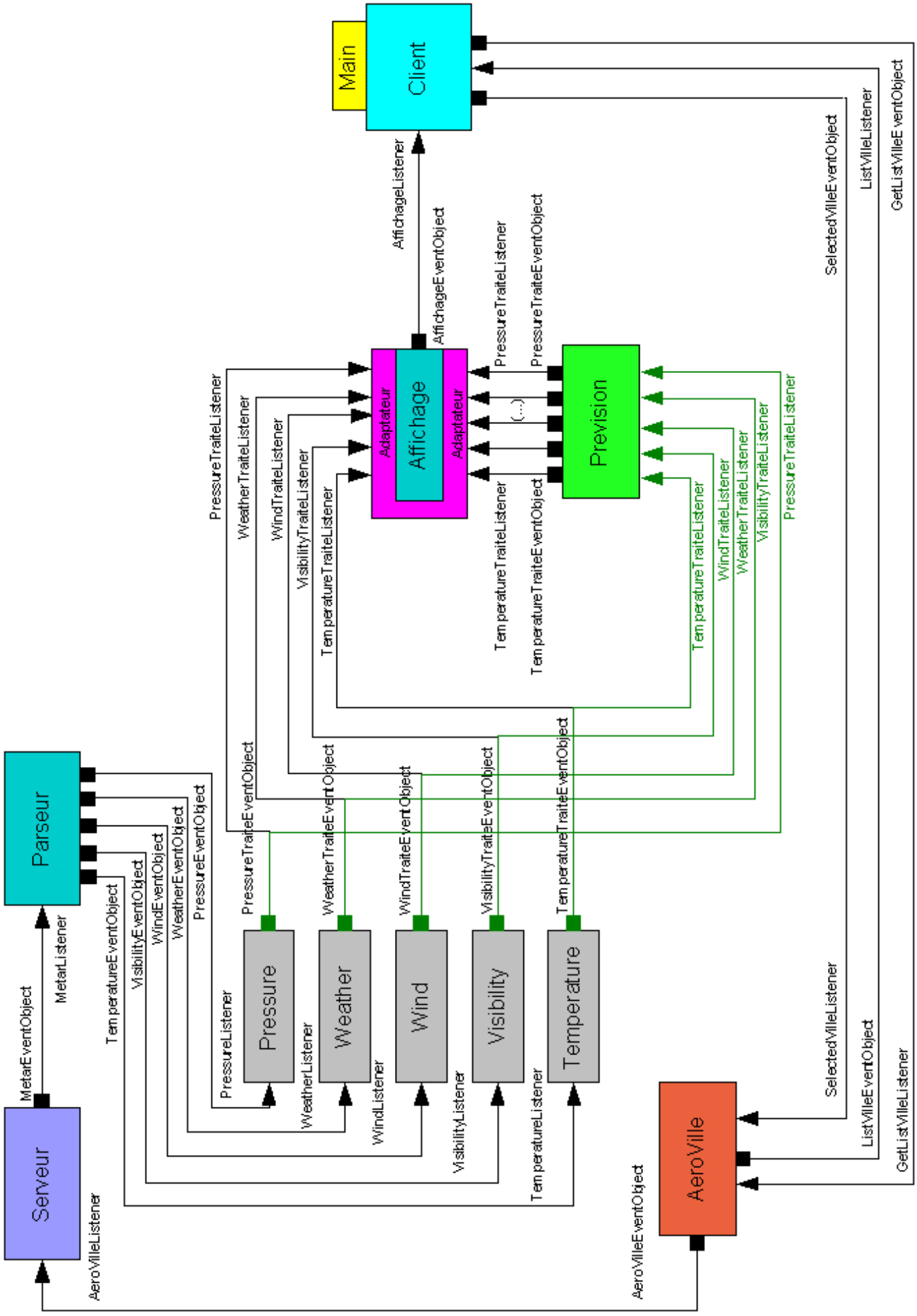
Le maillage des bornes météo étant assez lâche (3 pour la Bretagne à Rennes, Brest et Nantes), les villes proposées par l'application seront repérées par leurs distances aux trois stations météo les plus proches et les données météo affichées seront calculées par interpolation entre les données de ces trois bornes. Les règles d'interpolation ne sont pas les mêmes pour les différents types de données.

L'application Météo a été repensée dans cette deuxième partie de projet en termes de composants JavaBean.

2 Architecture de l'application Météo

2.1 Diagramme de composants

Représentation graphique de l'architecture de l'application météo en termes de composants JavaBeans.


Main
Client
AffichageListener
AffichageEventObject
Affichage
Adaptateur
Adaptateur
(...)
Prevision
PressureTraiteListener
PressureTraiteEventObject
WeatherTraiteListener
WindTraiteListener
VisibilityTraiteListener
TemperatureTraiteListener
SelectedVilleEventObject
ListVilleListener
GetListVilleEventObject
Parseur
MetarListener
MetarEventObject
PressureEventObject
WeatherEventObject
WindEventObject
VisibilityEventObject
TemperatureEventObject
PressureListener
WeatherListener
WindListener
VisibilityListener
TemperatureListener
Pressure
Weather
Wind
Visibility
Temperature
PressureTraiteEventObject
WeatherTraiteEventObject
WindTraiteEventObject
VisibilityTraiteEventObject
TemperatureTraiteEventObject
Serveur
AeroVilleListener
AeroVille
AeroVilleEventObject
SelectedVilleListener
ListVilleEventObject
GetListVilleListener

2.2 Description des différents composants

2.2.1 Le composant « Parseur »

Composant gérant le parsage des metars en éléments météo.

L'événement qu'il peut recevoir est :
- Metar : contient une liste de metars à parser

Les événements qu'il peut émettre sont :
- Visibility : contient l'information sur la visibilité
- Temperature : contient la température de rosé et la température
- Wind : contient les informations sur le vent
- Weather : contient l'information sur le temps
- Pressure : contient la pression atmosphérique

2.2.2 Le composant « AeroVille »

Ce composant gère les villes et les aéroports.

Les événements qu'il peut recevoir sont :
- GetListVille : demande de la liste des villes
- SelectedVille : contient la ville pour la recherche de l'aéroport

Les événements qu'il peut émettre sont :
- ListVille : contient la liste des villes
- AeroVille : contient la liste des codes d'aéroport et les distances

La propriété est :
- aeroVilleFile : fichier contenant les informations sur les villes et les aéroports.

2.2.3 Le composant « Serveur »

Ce composant envoie les metars au parseur.

L'événement qu'il peut recevoir est :
- Aeroville : contient la liste des codes d'aéroport et les distances

L'événement qu'il peut émettre est :
- Metar : contient une liste de metars

Les propriétés sont :
- url : url du serveur ou du fichier
- test : permet de savoir si on est en mode serveur test ou réel.

2.2.4 Le composant « Pressure »

Composant gérant la pression atmosphérique.

L'événement qu'il peut recevoir est :
- Pressure : contient l'information sur la pression atmosphérique

L'événement qu'il peut émettre est :
- PressureTraite : contient l'information sur la pression atmosphérique après traitement

2.2.5 Le composant « Wind »

Composant gérant le vent.

L'événement qu'il peut recevoir est :
- Wind : contient les informations sur le vent

L'événement qu'il peut émettre est :
- WindTraite : contient les informations sur le vent après traitement

2.2.6 Le composant « Weather »

Composant gérant le temps global.

L'événement qu'il peut recevoir est :
- Weather : contient l'information sur le temps

L'événement qu'il peut émettre est :
- WeatherTraite : contient l'information sur le temps après traitement

2.2.7 Le composant « Temperature »

Composant gérant les températures.

L'événement qu'il peut recevoir est :
- Temperature : contient la température et la température de rosé

L'événement qu'il peut émettre est :
- TemperatureTraite : contient les informations sur la température après traitement

2.2.8 Le composant « Visibility »

Composant gérant la visibilité.

L'événement qu'il peut recevoir est :
- Visibility : contient l'information sur la visibilité

L'événement qu'il peut émettre est :
- VisibilityTraite : contient l'information sur la visibilité après traitement

2.2.9 Le composant « Prevision »

Composant gérant les prévisions météorologiques.

Les événements qu'il peut recevoir sont :
- TemperatureTraite : contient les informations sur la température après traitement
- VisibilityTraite : contient l'information sur la visibilité après traitement
- WindTraite : contient les informations sur le vent après traitement
- WeatherTraite : contient l'information sur le temps après traitement
- PressureTraite : contient l'information sur la pression atmosphérique après traitement

Les événements qu'il peut émettre sont :
- TemperatureTraite : contient les informations sur la température après traitement
- VisibilityTraite : contient l'information sur la visibilité après traitement
- WindTraite : contient les informations sur le vent après traitement
- WeatherTraite : contient l'information sur le temps après traitement

- PressureTraite : contient l'information sur la pression atmosphérique après traitement

NB : Le composant prévision n'a pas été implémenté.

2.2.10 Le composant « Client »

Composant gérant l'interface graphique.

Les événements qu'il peut recevoir sont :
- ListVille : contenant la liste des villes.
- Affichage : contient le texte à afficher au client

Les événements qu'il peut émettre sont :
- SelectedVille : contient la ville pour la recherche de l'aéroport
- GetListVille : demande de la liste des villes

Ce composant contient le main, celui-ci instancie les composants, effectue les liaisons entre les différents composants et permet le lancement de l'application.

2.2.11 Le composant « Affichage »

Composant traitant les informations pour créer un texte qui pourra être afficher par le client.

Les événements qu'il peut recevoir sont :
- TemperatureTraite : contient les informations sur la température après traitement
- VisibilityTraite : contient l'information sur la visibilité après traitement
- WindTraite : contient les informations sur le vent après traitement
- WeatherTraite : contient l'information sur le temps après traitement
- PressureTraite : contient l'information sur la pression atmosphérique après traitement

L'événement qu'il peut émettre est :
- Affichage : contient le texte à afficher au client

La propriété est :
- adaptateur : adaptateur gère les arrivés d'événements.

On a utilisé un adaptateur pour pouvoir savoir d'où provient le texte contenant les informations météorologiques : des prévisions ou des informations météorologiques « normales ». Il trie les événements Affichage.

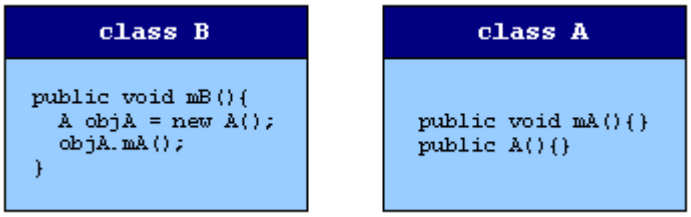
3 Réflexion sur l'approche par composants

La programmation par objets présente un certain nombre de limites, pour y remédier, on peut utiliser la programmation par composants.

3.1 *Limites de la programmation par objets*

- Il n'est généralement pas possible de réutiliser des classes sans en définir des nouvelles.
- L'héritage peut rompre l'encapsulation, c'est-à-dire que la classe fille devient dépendante de l'implémentation de la classe parente.
- On n'a pas une vue globale de l'architecture d'une application avec les objets.
- L'expression de besoins non-fonctionnels n'est pas possible (ex : persistance, performance…)
- Il existe une forte relation de couplage dans la programmation par objets.

Par exemple : Prenons 2 classes, une classe A et une classe B.

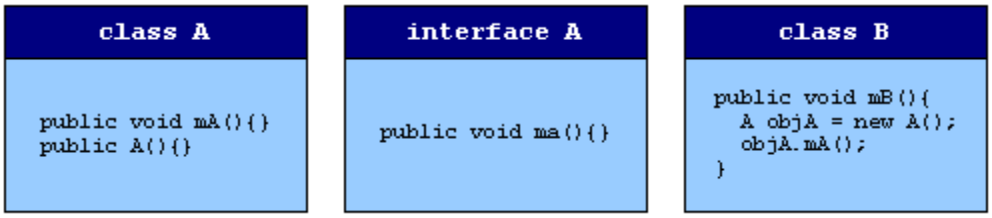


Problème : Comment savoir que B utilise A si l'on n'a pas accès au code de B ?

3.2 *La programmation par composants*

La programmation par composants permet d'apporter une solution au problème énoncé dans la partie précédente. Car on sait qu'elles sont les interfaces requissent par un composant et celles disponibles.

Un exemple simple :



On peut appeler la méthode *mA()* de *A* dans *B* sans problème car elle est fournie par l'interface *A*.
On a ici un couplage qui est faible.

La communication entre les différents composants se fait par des événements. En java, le modèle événement est tout en objet.

3.2.1 Les inconvénients de la programmation par composants

La création d'événement est longue et contraignante (en Java) car :

- Il faut créer sa classe événement qui hérite de java.util.EventObject
- Il faut créer l'interface listener qui hérite de java.util.EventListener
- Dans la classe source, il faut créer les méthodes pour inscrire les auditeurs
- Il faut gérer le fait de lancer un événement

Il y a des conventions de nommage à respecter.

Dans le cas où, il y a un auditeur et un événement, mais plusieurs sources et plusieurs réactions possible, il faut utiliser un adaptateur pour résoudre cette contrainte. Cela implique une recompilation lorsque l'on ajoute une nouvelle source.

Factoriser un logiciel en composants nécessite un important travail d'analyse. Cette factorisation n'est pas sans imposer quelques désagréments et arrachage de cheveux au chef de projet et aux développeurs participants à l'analyse. La rédaction des signatures des méthodes devra être particulièrement soignée, car modifier une signature nécessitera de retravailler toutes les portions de codes du projet qui font appel au composant, et l'on perdrait alors les bénéfices de l'indépendance des briques logicielles.

Le bénéfice de la programmation par composants se voit sur le long terme.

3.2.2 Les avantages de la programmation par composants

En faisant de l'introspection sur les composants avec *BeanInfo*, il est possible de connaitre le contenu d'un composant : sous composants, comportements, ports d'entrée et sortie, …

Les avantages que la programmation par composants présente lors d'un développement d'un projet sont les suivantes :

- L'équipe de développement peut-être divisée en sous-groupes, chacun se spécialisant dans le développement d'un composant.
- On peut faire sous-traiter le développement d'un composant, à condition d'en avoir bien réalisé les spécifications au préalable.
- La modification d'un composant ne nécessite pas la recompilation du projet complet.
- Lors de la mise à jour d'un composant, alors que le logiciel à déjà été livré au client, la livraison en est facilité, puisqu'il n'y a pas besoin de relivrer l'intégralité du projet, mais seulement le composant modifié.
- Il est possible de développer les différents composants dans des langages de programmation différents. Ainsi, un composant nécessitant une fonctionnalité particulière pourra profiter de la puissance d'un langage dans un domaine particulier, sans que cela n'influe sur l'ensemble du développement du projet.
- La réutilisabilité d'un composant permet un gain de productivité non négligeable car elle diminue le temps de développement, d'autant plus que le composant est réutilisé souvent.