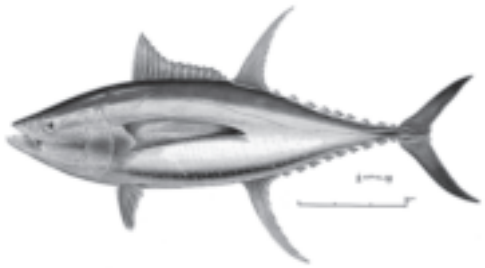
2e partie
Un logiciel d'extraction des données environnementales

Martin Desruisseaux,
Michel Petit, Laure Gardel



Thunnus albacares (Bonnaterre, 1788)

Le prototype SEASview de SIO (Système d'information océanique) présenté ici est le résultat de plusieurs années d'étude de la station SEAS-Réunion. SEASview a été adapté aux besoins spécifiques de l'Ifremer, l'utilisateur peut afficher, à sa convenance, pour une échelle spatio-temporelle donnée, les différents paramètres océanographiques et halieutiques au sein de cartes géoréférencées sans se soucier des divers fichiers des données de base et des systèmes d'intégration nécessaires.

Le projet Palangre vise entre autres à faciliter les accès à une base de données comprenant l'ensemble des données satellitaires fournies par l'IRD et des données d'abondance fournies par l'Ifremer.

Le prototype SEASview comprend une interface de programmation d'application, API[1], pour les développeurs, sur laquelle est construite une interface graphique pour le reste des utilisateurs. Ses fonctionnalités s'apparentent aux fonctions les plus basiques trouvées dans plusieurs systèmes d'information géographiques (SIG). Le logiciel présenté ici se distingue toutefois des logiciels commerciaux sur trois aspects :

– Il met davantage l'accent sur la navigation dans le temps. La plupart des SIG commerciaux sont conçus pour le domaine terrestre, dans lequel les images sont très contrastées mais ne varient pas beaucoup d'une journée à l'autre.

1. *API*

Le paysage océanographique en revanche est plus nuancé et se transforme rapidement. Une telle dynamique n'est pas très bien gérée par la plupart des outils du domaine terrestre.

– Il représente les données comme des matrices de paramètres environnementaux quantifiables (température, concentration en chlorophylle-*a*, etc.) auxquels s'ajoutent quelques catégories qualitatives (nuage, terre, absence de données, etc.). La plupart des SIG commerciaux savent bien gérer séparément les paramètres quantitatifs et les catégories qualitatives, mais savent moins bien travailler sur des couches qui combinent les deux.

– Il contient quelques facilités pour croiser les données environnementales avec les données de pêches (chlorophylle-*a* aux positions de pêches, etc.).

Les images apparaissant dans l'explorateur sont le fruit de mesures satellitaires et de calculs expliqués dans les chapitres précédents. Dans tous les aspects de la navigation (sélection de la région géographique, extraction des valeurs, etc.), les valeurs des paramètres géophysiques et les coordonnées sont exprimées en unités du monde réel, jamais en indices pixels. Dans le cas des coordonnées, les unités sont les degrés de longitude et de latitude sur l'ellipsoïde WGS 1984.

Installation de l'environnement Java et configuration des sources ODBC

Le logiciel peut être exécuté sur n'importe quelle plate-forme disposant d'un environnement Java récent (Windows, Linux, Solaris, Mac OS-X…), d'un logiciel de base de données (Access, Oracle, MySQL, PostgreSQL…) et de 128 Mo de mémoire vive. L'environnement Java doit comprendre au minimum les deux modules suivants :

- *jBEta* (JRE) version 1.4 ou ultérieure ;
- *jAdig* (JAI) version 1.1 ou ultérieure.

Les programmes d'installation sont fournis sur le cédérom SEASview, dans le répertoire install. Deux ensembles de programmes sont proposés : un ensemble pour client (plus compact mais ne comprenant pas les outils de développement) et un ensemble plus volumineux pour développeur. Il faut n'installer qu'un seul de ces deux ensembles ; l'ensemble pour développeur inclut déjà tout l'ensemble pour client. Les deux tableaux suivants énumèrent les programmes d'installation à exécuter dans l'ordre, selon l'ensemble choisi. L'ensemble pour développeur est nécessaire à toute la partie API (cf. *fi*).

Une fois l'installation terminée, il est prudent de vérifier qu'elle s'est correctement effectuée. Vous pouvez faire une première vérification en tapant l'instruction suivante sur la ligne de commande (par exemple dans une fenêtre MS-DOS) :

```
java –version
```

▽ [illegible] 6

[illegible]

([illegible])

Fichiers d'installation (.exe)	Adresse internet
`j2re-1_4_0-win-i`	http://java.sun.com/j2se/1.4/jre/
`jai-1_1_1-lib-win-jre`	http://java.sun.com/products/java-media/jai/downloads/download.html dans la section « JRE Install for Windows, Solaris SPARC, Solaris x 86, and Linux »

▽ [illegible] 7

[illegible]

([illegible])

Fichiers d'installation (.exe)	Adresse internet
`j2sdk-1_4_0-win`	http://java.sun.com/j2se/1.4/
`jai-1_1_1-lib-win-jdk`	http://java.sun.com/products/java-media/jai/downloads/download.html dans la section « JDK Install for Windows, Solaris SPARC, Solaris x 86, and Linux »
`NetBeans-release321`	http://www.netbeans.org/downloads.html
Documentation (.zip)	**Adresse internet**
`j2sdk-1_4_0-doc`	http://java.sun.com/j2se/1.4/
`jai-1_1-pdf1-doc`	http://java.sun.com/products/java-media/jai/docs/index.html

Vous devriez obtenir un affichage ressemblant aux lignes suivantes :

java version 1.4.0
Java (TM) 2 Runtime Environment, Standard Edition (build 1.4.0)
Java HotSpot (TM) Client VM (build 1.4.0-beta3-b84, mixed mode)

L'affichage peut varier. L'exemple ci-dessus était pour le Java de Sun Microsystems. Le Java d'IBM donne un affichage légèrement différent, mais qui est tout aussi correct s'il indique au moins la version 1.4 (sur la première ligne). L'application du projet Palangre ne demande pas d'installation en tant que tel. Il suffit de copier le contenu du répertoire install/Palangre à n'importe quel endroit du disque dur. Ce répertoire doit contenir les fichiers suivants :

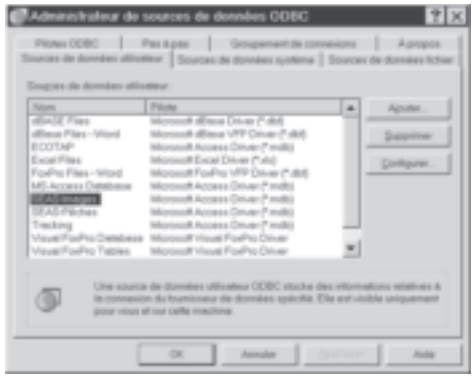
`Palangre.jar` : Le code compilé de l'application Palangre.
`Images.mdb` : La base de données d'images.
`Pêches.mdb` : La base de données des pêches.

Déclaration des bases de données comme source de données ODBC

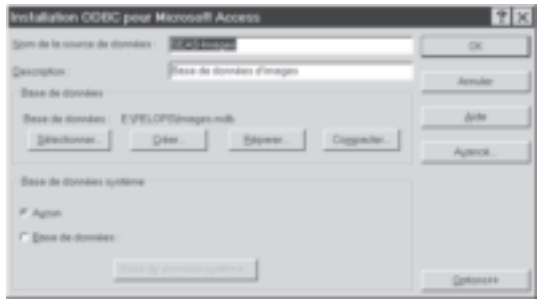
Le logiciel nécessite deux bases de données : une pour les données satellitaires et une pour les données de pêche. Leur structure sera décrite ci-après. Dans

la version actuelle, ces bases de données ont été construites avec le logiciel Microsoft Access sous Windows[2] et doivent être déclarées comme *Source de données ODBC* dans le panneau de configuration de Windows.

1) Ouvrez la fenêtre d'administration des sources de données ODBC :
Démarrer
Paramètres
Panneau de configuration
Source de données ODBC



2) Ouvrez la fenêtre d'ajout d'une source de données :
Ajouter
Microsoft Access driver
Terminer



1) Ouvrez la fenêtre d'administration des sources de données ODBC
   Démarrer
   Paramètres
   Panneau de configuration
   Source de données ODBC
2) Ouvrez la fenêtre d'ajout d'une source de données
   Ajouter
   Microsoft Access driver
   Terminer

Dans le champ **Nom de la source de données**, il faut entrer **SEAS-Images** (en respectant les majuscules et les minuscules). Le contenu du champ Description est laissé libre pour y entrer les commentaires de son choix. Cliquez ensuite sur **Sélectionner** et sélectionnez le fichier **Images.mdb** (sa position sur le disque dur dépend de l'endroit où il se trouve). Cliquez enfin sur **Ok**. La même procédure

2. Nous allons probablement abandonner Microsoft Access dans un avenir proche en faveur d'un autre logiciel de base de données, peut-être MySQL ou PostgreSQL.

doit être répétée pour la base des données de pêches (**Pêches.mdb**). Elle doit être déclarée sous le nom **SEAS-Pêches**.

Configuration de l'application Java

L'application du projet Palangre doit être configurée avant qu'elle ne soit utilisable. Cette configuration consiste surtout à lui spécifier le répertoire racine des images. À partir d'une fenêtre MS-DOS, entrez les commandes suivantes en remplaçant `[path]` par chemin de l'application :

```
set classpath = [path]\Palangre.jar
java fr.ird.sql.ControlPanel
```

Le tableau 19 devrait apparaître. Les informations qu'il contient sont appropriées pour les bases de données déclarées ci-dessous (Structure de la base de données satellitaires), sauf le répertoire racine des images. Remplacez ce dernier par celui qui convient pour votre système (exemple : `C :/Projets/Palangre/Images`) et confirmez avec le bouton **Ok**. Vous pourrez vérifier si la connexion avec la base de données fonctionne en entrant la commande suivante :

```
java fr.ird.sql.image.ImageDataBase -browse
```

Structure des bases de données

Les données de pêches et les informations sur les données satellitaires apparaissent dans deux bases de données disjointes, généralement maintenues par des organismes distincts (l'Ifremer et l'IRD) et utilisables indépendamment. Ces bases de données nous servent à la fois :

– de catalogues pour recenser les données disponibles et identifier les trous ;

– de moteurs de recherche pour retrouver rapidement des images ou des données de pêches à des coordonnées spatio-temporelles arbitraires ;

– d'organisateurs pour construire à la volée des tableaux croisés entre les données de pêches et les données environnementales ;

– d'enregistreurs pour recevoir les résultats de calculs coûteux ou laborieux.

La base de données satellitaires ne contient pas les données environnementales elles-mêmes, mais plutôt des références vers les fichiers qui contiennent ces données. On pourrait plus pertinemment la qualifier de base de méta-données satellitaires.

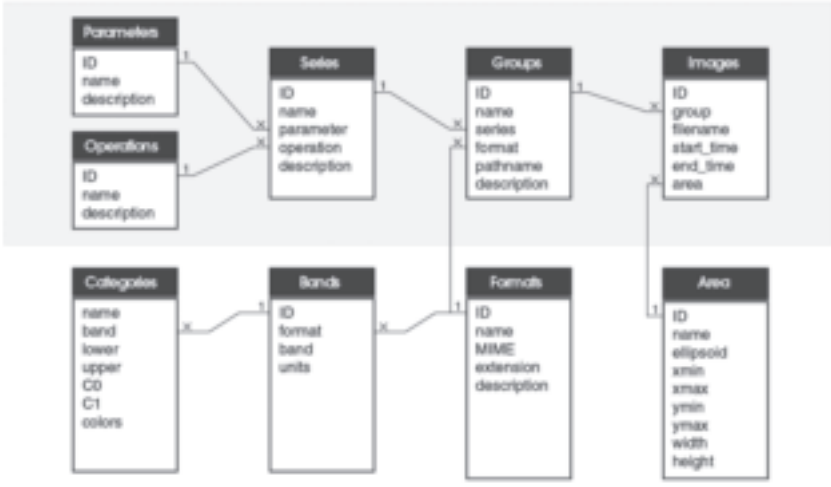
Structure de la base de données satellitaires

La base de données doit pouvoir contenir des informations sur les données de différents paramètres (température, vorticité, altimétrie, concentrations en chlorophylle-*a*…) enregistrées dans différents formats d'image, sur différents cédéroms et couvrant différentes régions géographiques. Il fallait la structurer de façon à couvrir le plus grand champ d'applications possibles sans déboucher sur un système trop compliqué. La structure retenue est bien adaptée à nos données, mais ne saurait couvrir tous les besoins. Les personnes désirant une structure beaucoup plus puissante (quitte à en payer le prix sur le plan de la complexité) sont invitées à consulter les propositions du consortium OpenGIS[3].

La figure 121 illustre la structure de notre base de données. Il s'agit du minimum que s'attendra à trouver le programme Java qui l'utilisera. L'utilisateur ne devrait pas supprimer ou renommer des champs qui appartiennent à cette structure minimale. Il peut toutefois ajouter de nouveaux champs, par exemple le nom d'une personne responsable pour chaque série.

Comme toute base de données, celle-ci est composée de plusieurs tables. La table **Images** constitue le but ultime de cette base de données. Chaque image appartient (indirectement) à une série de la table **Series**, qui représente elle-même un paramètre et une opération des tables **Parameters** et **Operations**. Les tables illustrées en dessous (**Areas**, **Formats**, **Bands** et **Categories**) apportent des informations nécessaires à l'interprétation des données mais n'interviennent pas dans les regroupements des images en catégories. La liste suivante (tabl. 20) résume les fonctions de chaque table.

3. *In* [illegible] dans http://www.opengis.org/techno/specs.htm. Nous ne nous sommes pas rapprochés davantage des spécifications d'OpenGIS, le développement d'outils GIS étant à la limite du cadre du projet Palangre. Toutefois, le respect des normes OpenGIS est un objectif majeur de tous développements futurs.



▽ *Fig I 21*

méta-

▽ *E20*

Parameters	Liste des paramètres géophysiques (, , , etc.) que représentent les images.
Operations	Liste des opérations (, , , etc.) qui ont servi à produire les images. Cette table complète la table des paramètres pour le regroupement des images.
Series	Combinaison d'un paramètre et d'une opération, ainsi que d'autres critères laissés à la discrétion de l'utilisateur. Par exemple, la combinaison du paramètre avec l'opération , à laquelle on ajoute le critère qu'il s'agit d'images acquises par l'antenne de l'île de la Réunion, donne la série . Cette table serait un endroit approprié où ajouter des informations sur le capteur ou sur l'origine des images.
Groups	Les images d'une série se répartissent parfois dans plusieurs sous-groupes. La définition d'un groupe est laissée à la discrétion de l'utilisateur. Souvent, il s'agit du numéro du cédérom des images. On prend ainsi en compte le fait que les images d'une série peuvent être distribuées sur plusieurs cédéroms. Un groupe pourrait aussi correspondre à une certaine version des images. Notons que c'est à ce niveau seulement qu'apparaît la notion de fichier (son format et sa localisation). Toutes les tables précédentes faisaient abstraction de tout concept de fichiers.
Images	Liste de toutes les images répertoriées. À partir d'une image on peut remonter à son groupe, puis à sa série et enfin au paramètre qu'elle représente. Pour chaque image, nous spécifions une date de début et une date de fin d'acquisition. Pour les images de moyennes ou de synthèses, il s'agira des dates de la première et dernière image ayant servi à calculer la moyenne ou synthèse. Pour les images instantanées dont nous ignorons l'heure exacte de l'acquisition, il s'agira des dates d'une plage de temps (souvent 24 heures) qui contient l'heure à laquelle fut acquise l'image.
Areas	Coordonnées géographiques des régions couvertes par les images. Cette table complète la table **Images**. La version actuelle ne comprend pas encore d'informations sur les projections cartographiques utilisées, à part l'ellipsoïde.

Formats	Format des images (PCX, GIF, etc.). Cette table peut contenir des informations qui ne sont pas comprises dans le nom du fichier, par exemple le nombre de bits par données dans les fichiers binaires.
Bands	Liste des bandes apparaissant dans chacun des formats. La plupart des images du projet Palangre n'ont qu'une seule bande. Les données altimétriques constituent la principale exception, puisqu'elles forment des images à trois bandes.
Categories	Liste des catégories apparaissant dans chacune des bandes. Cette table indique comment interpréter certaines plages de pixels (**t** , **g** , etc.). Si une plage de pixels représente les valeurs d'un paramètre géophysique, alors cette table donne les coefficients C_i de l'équation $y = C_0 + C_1x$ qui serviront à convertir les pixels en valeurs.

Mise à jour de la base de données satellitaires

Le logiciel Access n'est pas indispensable pour utiliser la base de données avec l'application Java. Son interface graphique se révèle toutefois fort utile pour les mises à jour, notamment l'ajout de nouvelles images. À l'ouverture, la base de données devrait présenter la figure 122.

La table **Series** constitue un bon exemple des liens qui existent entre les tables. Deux de ces colonnes offrent des listes déroulantes qui se réfèrent au contenu des tables **Parameters** et **Operations** (fig. 123). Avant de créer une nouvelle série, il faut d'abord vérifier si les paramètres et opérations d'intérêt figurent dans leurs tables respectives et les ajouter au besoin.

Chaque série peut contenir plusieurs groupes, pour différencier par exemple les images à haute résolution de leur aperçu. Un symbole d'expansion [+] devant chaque ligne permet d'afficher la liste des groupes pour une série (fig. 124).

Structure de la base de données des pêches

La base des données de pêches, fournie par l'Ifremer, ne contient initialement qu'une seule table. Cette table (**Captures**) comporte les coordonnées spatio-temporelles des filages ainsi que le nombre de prises par espèces. Pour le projet Palangre, nous avons ajouté quelques champs supplémentaires à la table des captures et créé deux autres tables. La figure 125 représente les tables de cette base de données ainsi que quelques-uns de leurs champs.

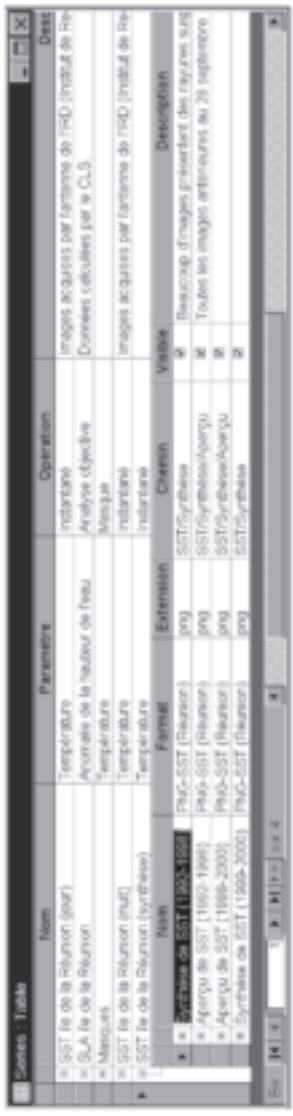
La table des espèces sert à traduire les codes de la FAO (par exemple SWO pour l'espadon, ALB pour le germon, etc.). Cette table doit énumérer au moins toutes les espèces qui apparaissent dans la table des captures, car le programme Java utilise cette liste pour distinguer les champs de la table **Captures** qui correspondent effectivement à des prises.

Le tableau 21 donne les principaux champs de la table des captures. Nous n'énumérons pas tous les champs correspondant aux prises par espèce. Pour nos besoins d'analyse, nous avons calculé et rajouté les champs **distance**, **distance_pêche** et **distance_côte**.

La table **Environnements** contient une sélection de paramètres géophysiques (température, anomalie de la hauteur de l'eau, etc.) associés aux pêches. Les paramètres calculés comprennent :

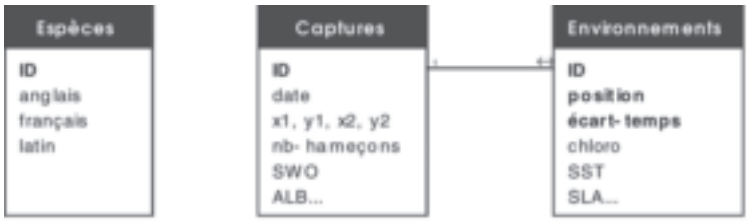
▽ Fig 22 – Fig

▽ Fig 23 – Fig



▽ Fig 24 – Fig

– la concentration en chlorophylle-*a* (mg.m^{-3}) ;
– la température de surface (°C) ;
– l'anomalie de hauteur de l'eau (cm) ;
– les composantes U et V du courant géostrophique (cm.s^{-1}) ;
– une estimation du pompage d'Ekman dû au vent (cm.j^{-1}) ;
– la profondeur du plancher océanique (m) ;
– une indication des gradients pour chacun des paramètres précédemment cités.



▽ *Fig 125*

▽ *Tabl 121*

date	La date du filage (sans indication de l'heure).
x1, y1, x2, y2	Les coordonnées de début et de fin du filage en degrés fractionnaires. Nous n'avons pas à prime abord d'indication sur la trajectoire de la ligne entre ces deux points.
droite	Une valeur booléenne (vrai/faux) indiquant si la ligne a probablement été filée en ligne droite. Cette information a été calculée par l'ifremer en estimant la longueur de la ligne à partir de la durée du filage et en comparant cette valeur à la distance entre les points de début et de fin. Les résultats ont été incorporés dans la base de données par l'IRD.
distance	La distance orthodromique entre les points de début et de fin du filage. Cette distance serait approximativement égale à la longueur de la ligne si celle-ci était filée en ligne droite (en négligeant ses variations de profondeur).
distance_pêche	La distance orthodromique la plus courte entre cette ligne et les autres lignes filées le même jour. La distance est mesurée à partir du milieu du filage, c'est-à-dire la moyenne des positions de début et de fin.
distance_côte	La distance orthodromique la plus courte entre cette ligne et la côte (incluant l'île de la Réunion, Madagascar et la côte africaine).
nb_hameçons	Le nombre d'hameçons sur la ligne de pêche.
SWO, ALB…	Le nombre d'individus capturés pour chaque espèce. La table contient de nombreux champs, mais les principaux sont SWO, ALB, YFT et BET. Ces sigles correspondent aux codes de la FAO pour l'espadon, le germon, l'albacore et le thon obèse.

Les gradients ont été estimés à l'aide de l'opération `GradientMagnitude` de la bibliothèque *ITK* (voir ci-dessous). Ce dernier estime d'abord le gradient zonal et le gradient longitudinal à l'aide d'opérateurs de Sobel, puis calcule le module du gradient avec $\sqrt{(x^2 + y^2)}$.

Chacun des paramètres précités et leurs gradients apparaissent dans leur propre champ (colonne) de la table des données environnementales (**Environnements**). Chacune des lignes de cette table commence par trois colonnes qui font le lien avec la table des captures (tabl. 22).

Tous les paramètres environnementaux sélectionnés ont été évalués à cinq positions sur la ligne de pêche : au début (0 %), à la fin (100 %), au milieu (50 %), au quart (25 %) et au trois quarts (75 %). Les valeurs au début et à la fin sont les plus fiables ; les autres supposent que la ligne a été filée en ligne droite. Tous ces paramètres à chacune des cinq positions ont été eux-mêmes estimés avec

▽ Table 22

ID	L'identification du filage pour lequel les paramètres environnementaux ont été évalués. Ce numéro se réfère au champ **ID** de la table **Captures**.
position	La position sur la ligne de pêche à laquelle les paramètres ont été évalués. 0 désigne la position du début, 100 la position de fin et les autres valeurs entre 0 et 100 représentent des positions intermédiaires. Les valeurs supérieures à 100 peuvent représenter d'autres régions arbitraires (par exemple, une moyenne dans un rayon de 10 milles nautiques).
écart_temps	L'écart de temps (en nombre de jours) entre le moment de la pêche et le moment où les paramètres environnementaux ont été évalués. La valeur 0 signifie que les paramètres ont été évalués le jour même. La valeur - 5 (par exemple) signifie qu'ils ont été évalués 5 jours avant.

cinq écarts de temps différents : le jour de la pêche, cinq, dix et quinze jours avant ainsi que cinq jours après. Le tout forme un ensemble de plusieurs centaines de variables environnementales par ligne de pêche.

Aperçu de l'interface de programmation (API)

L'exploitation des données fait intervenir des accès aux bases de données, des positionnements géographiques, des traitements d'images, des analyses statistiques, de la modélisation par réseaux de neurones artificiels ainsi que de la programmation pour automatiser le tout. L'industrie informatique a standardisé les accès aux bases de données depuis plusieurs années, grâce au langage SQL[4]. Les positionnements géographiques et les traitements d'images sont aussi en voie de standardisation (notamment par le consortium OpenGIS), mais les efforts en ce sens sont beaucoup plus récents et peu intégrés dans les logiciels commerciaux actuels. Quant aux analyses statistiques et réseaux de neurones, ils ne sont pas à notre connaissance l'objet de standards informatiques.

Cet API, développé pour le croisement des données, s'appuie sur les standards précédemment cités et sert lui-même de base à une application graphique. Le tableau 23 illustre les dépendances entre les principaux modules.

▽ Table 23

Interface graphique utilisateur **Moteur de croisement des données**		
SQL (via JDBC)	OpenGIS	Java Advanced Imaging

4.

Les interfaces utilisateurs sont souvent des goulots d'étranglement, à la fois pour l'énergie qu'exigent leurs développements ainsi que pour les fonctionnalités forcément réduites qu'elles offrent à l'utilisateur. Ceux qui voudront tenter de nouveaux types de croisements risquent de subir les limites de l'interface utilisateur, limites dont seul un accès à l'API de programmation permet de s'affranchir.

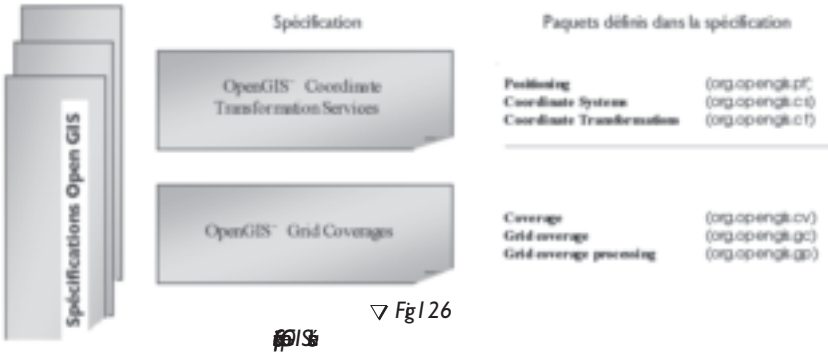
Introduction aux spécifications OpenGIS

Le consortium OpenGIS publie un grand nombre de spécifications couvrant un vaste champ d'applications : géométrie, imagerie, échange d'informations à travers les réseaux, etc. Le respect de ces spécifications permet à des groupes de recherche *ap* indépendants d'échanger plus facilement des informations géographiques, même s'ils utilisent des systèmes différents. Le consortium résume ainsi ses objectifs dans son site officiel :

Se familiariser avec les spécifications d'OpenGIS et les appliquer ensuite, demandent un investissement de temps considérable à court terme. La difficulté vient principalement de leur caractère récent. Peu de solutions industrielles ont eu le temps d'intégrer ces spécifications à la fin de 2001, ce qui nous obligea à implémenter nous-même les modules dont nous avions besoin. Nous croyons toutefois que c'est une approche profitable à plus long terme, puisqu'elle offrira vraisemblablement des possibilités accrues d'interactions avec les logiciels industriels du futur.

Les spécifications d'OpenGIS se divisent en un ensemble de documents (plus de 15), chaque document contenant lui-même un ensemble de « paquets ». La figure 126 illustre les deux spécifications que nous avons implémentées, ainsi que leurs paquets. Les paquets sont illustrés dans l'ordre de leurs dépendances. Par exemple, le paquet *Gid* a besoin de *Cg* , qui a lui-même besoin de *Cti* et ainsi de suite jusqu'à *g* , le seul paquet réellement autonome.

Chacun de ces paquets contient plusieurs classes, qui contiennent elles-mêmes plusieurs méthodes. Pour une liste exhaustive, on se réfèrera au site officiel



▽ Fig I 26

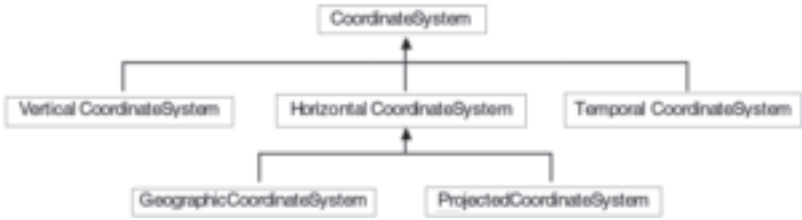
d'OpenGIS (www.opengis.org). Nous n'introduisons ici que les classes les plus importantes pour notre étude. Le lecteur averti notera que certains éléments dans les sections suivantes diffèrent légèrement de la syntaxe officielle d'OpenGIS. Ces différences résultent d'un compromis entre le respect de la spécification OpenGIS (qui se veut neutre quant au langage de programmation) et le respect des standards existants en Java. Dans les cas où ne peut être toléré aucun écart vis-à-vis de la norme OpenGIS (par exemple lors d'interactions avec des logiciels commerciaux), une série d'adaptateurs permet de faire les conversions dans les deux sens entre nos classes modifiées et les classes strictement OpenGIS. Nos classes modifiées ont pour racine le paquet `net.seas.opengis`, alors que les classes officielles restent dans `org.opengis`.

Systèmes de coordonnées (paquet `opengis.cs`)

Tous les systèmes de coordonnées sont représentés par une sous-classe de `CoordinateSystem`. Chacune de ces sous-classes représente un système de coordonnées simple. Par exemple, la sous-classe `VerticalCoordinateSystem` représente un système avec une seule dimension le long de l'axe vertical, tandis que `HorizontalCoordinateSystem` représente un système à deux dimensions dans le plan horizontal. Un système de coordonnées arbitraire peut être construit par l'assemblage de deux ou plusieurs autres systèmes de coordonnées. Par exemple, on peut construire un système à trois dimensions en assemblant `HorizontalCoordinateSystem` et `VerticalCoordinateSystem`. La figure 127 illustre la hiérarchie des principales classes.

Une multitude de combinaisons est possible, du fait qu'il existe plusieurs systèmes de coordonnées de chaque type (vertical, horizontal et temporel). Par exemple, un système vertical qui mesure les hauteurs avec un baromètre n'est pas identique à un système qui mesure les hauteurs avec un GPS[5]. Un système temporel utilisant l'heure UTC n'est pas identique à un autre système utilisant

5. Gps



▽ *Fig 127*

l'heure GMT[6]. Enfin, deux systèmes horizontaux peuvent différer par l'ellipsoïde de référence qu'ils utilisent. Ces détails sont précisés par un ensemble de classes secondaires (`Datum, Ellipsoid, Projection`, etc.).

Dans le cadre du projet Palangre, nous positionnons les données satellitaires dans un système de coordonnées à trois dimensions, soit dans l'ordre la longitude, la latitude et le temps. Il s'agit donc d'un assemblage de `GeographicCoordinateSystem` avec `TemporalCoordinateSystem`. Les angles sont mesurés en degrés sur l'ellipsoïde WGS 1984[7], tandis que le temps est mesuré en jours écoulés depuis le 1er janvier 1950 à 00 h 00 UTC. Si des transformations de coordonnées sont nécessaires, elles seront effectuées à la volée. Cela s'applique notamment à l'axe du temps, puisque les bases de données et le Java ont déjà leur propre type Date.

Transformations de coordonnées (paquet `opengis.ct`)

Chaque système de coordonnées (`CoordinateSystem`) ignore l'existence des autres. Les transformations sont effectuées par une autre classe, `CoordinateTransform`[8], qui sert d'intermédiaire entre deux `CoordinateSystems`. Les transformations sont obtenues indirectement, en utilisant encore une autre classe (`CoordinateTransformFactory`) qui sert en quelque sorte d'usine à fabriquer des transformations. La procédure à suivre à cet effet est la suivante :

– Obtenir un premier objet `CoordinateSystem` représentant le système d'origine (`source`).

6. L'heure UTC (Universal Time Clock) est basée sur une horloge atomique, tandis que l'heure GMT (Greenwich Mean Time) est basée sur des observations astronomiques. L'heure GMT ralentit et s'accélère d'une manière compliquée en fonction de l'orbite terrestre, tandis que l'heure UTC est régulière. De temps à autre, l'heure UTC est synchronisée sur l'heure GMT par l'introduction d'une seconde supplémentaire à la fin de certaines années (par exemple, la dernière minute de l'année 1995 était longue de 61 secondes). Les systèmes GPS sont synchronisés sur l'heure UTC, mais sans la seconde supplémentaire ajoutée à la fin de certaines années (de sorte que l'heure des systèmes GPS s'écarte progressivement de l'heure UTC).

7. Sauf les données dérivées de l'altimétrie, qui utilisent leur propre ellipsoïde. Toutefois, l'ellipsoïde utilisé pour les données de Topex/Poséidon étant très proche de l'ellipsoïde WGS 1984, la version actuelle du logiciel ignore la différence.

8. Les classes décrites dans cette section diffèrent légèrement de la spécification d'OpenGIS. La principale différence concerne la classe CoordinateTransform, que nous avons positionnée comme une sous-classe de `MathTransform` et renommée pour refléter ce changement (le nom utilisé dans la spécification d'OpenGIS était `CoordinateTransformation`). Mais le principe reste identique.

– Obtenir un second objet `CoordinateSystem` représentant le système désiré (`target`).

– Donner ces deux systèmes de coordonnées à `CoordinateTransformFactory`[9]. Cette dernière les analysera et, si elle détermine qu'une transformation est possible, produira un objet `CoordinateTransform` qui permettra de passer du système source vers le système `target`.

– Utiliser l'objet `CoordinateTransform` obtenu pour transformer autant de coordonnées que l'on veut. La transformation se fera toujours entre les deux systèmes de coordonnées précités. Pour changer de transformation, il faut recommencer la procédure.

Le développeur trouvera les détails de ces opérations dans la documentation *javadoc* (un ensemble de pages HTML). Dans le cadre du projet Palangre toutefois, il aura rarement besoin d'effectuer lui-même des transformations de coordonnées.

Matrices de données (paquet `opengis.gc`)

Toutes les données représentables sur une grille régulière (par exemple, une image) sont gérées par la classe `GridCoverage`. Les données qui ne sont pas nécessairement distribuées sur une grille régulière sont représentées par la classe plus générale Coverage, que l'on pourrait traduire par « couverture » au sens large. Les couvertures, régulières ou pas, possèdent toutes au moins les propriétés suivantes :

– Un système de coordonnées (`CoordinateSystem`) qui s'applique à l'enveloppe (voir le point suivant) ainsi qu'à toutes les coordonnées utilisées avec cette couverture. Le développeur ne travaillant qu'avec les données du projet Palangre peut s'épargner la peine de vérifier le système de coordonnées de chaque couverture, puisque nous utilisons un système assez uniforme (voir ci-dessous). Mais il ne pourra plus ignorer cette information si les données du projet Palangre sont croisées avec encore d'autres sources de données.

– Une enveloppe (`Envelope`) qui donne les coordonnées extrêmes de la couverture. Cette enveloppe doit avoir le même nombre de dimensions que le système de coordonnées. Dans le projet Palangre, chaque couverture (ou image) possède une enveloppe à trois dimensions qui donne dans l'ordre : la longitude minimale et maximale, la latitude minimale et maximale, ainsi que la date de début et de fin de l'image.

– Au moins une bande (`SampleDimension`[10]), parfois plusieurs. Chaque bande contient une mesure géophysique (par exemple, la température) à la position de chaque pixel. Les images du projet Palangre n'ont chacune qu'une seule bande, sauf les images dérivées des données altimétriques qui en ont trois (l'anomalie de la hauteur de l'eau ainsi que les composantes U et V du courant géostrophique).

9. Plus spécifiquement à sa méthode createFromCoordinateSystems(…).

10. OpenGIS préfère le terme *path* plutôt que *band* parce que le terme *band* se réfère trop spécifiquement à l'imagerie satellitaire, alors que les données pourraient avoir une autre origine (par exemple, un modèle numérique).

La classe `Coverage` (dont hérite `GridCoverage`) dispose de méthodes `evaluate` qui permettent d'obtenir la valeur des paramètres géophysiques à une coordonnée quelconque. À chaque appel d'une méthode `evaluate`, les valeurs sont calculées pour toutes les bandes et retournées dans un tableau.

Si l'image n'a qu'une seule bande, la longueur de ce tableau sera instanciée 1. L'encadré ci-après est extrait de la documentation [illegible] qui donne la signature de la méthode ainsi que sa spécification.

evaluate

evaluate *(Coordinate[illegible])*

Parameters : *coord-* [illegible]
dest- [illegible]
`getSampleDimensions ().size ()`

Returns : [illegible] `dest` [illegible] `dest`

Throws : `PointOutsideCoverageException` [illegible] `coord`

Pour le projet Palangre, la méthode `evaluate` retourne toujours les valeurs géophysiques des paramètres (par exemple, une température en degrés Celsius), jamais les valeurs entières des pixels. Les conversions sont faites à la volée. Si la donnée est manquante (par exemple, si la coordonnée spécifiée tombe sur un nuage ou sur la terre ferme), alors la valeur retournée est NaN ([illegible]). Les valeurs NaN du Java ont plusieurs propriétés utiles pour les traitements d'images, parmi lesquelles :

– Toute opération arithmétique faisant intervenir une valeur NaN produit un résultat NaN. Ainsi, il est possible d'appliquer des algorithmes déjà existants sur une image sans crainte d'obtenir des pixels aberrants, résultant par exemple de l'addition d'une valeur de température avec un nuage. Si une des données était NaN (le nuage), le résultat sera automatiquement NaN quelle que soit la valeur de température.

– Plusieurs valeurs NaN sont possibles. On peut avoir une valeur NaN pour les nuages et une autre pour la terre. Il est ainsi possible de continuer à distinguer les nuages de la terre même si les images résultent d'un calcul (par exemple, une convolution).

Notre logiciel effectue les conversions des valeurs de pixels en valeurs géophysiques et inversement, grâce à une liste de `Category` gérée par la classe `CategoryList`. Ce sont ces mêmes classes qui permettent à notre logiciel de s'y retrouver parmi les différentes valeurs NaN. La création de ces deux classes est l'ajout le plus significatif que nous ayons fait par rapport à la spécification d'OpenGIS. La spécification officielle d'OpenGIS prévoit en effet la répartition des valeurs de pixels en catégories, mais ne représente ces catégories que par du texte (par exemple forêt, zone urbaine, etc.), dans la plus pure tradition des

SIG commerciaux dédiés au domaine terrestre. Pour le milieu océanique, cela ne suffit pas. Nos images mélangent à la fois des catégories *qualitatives* (nuage, terre, etc.) et des catégories *quantitatives* (température, etc.). Il nous faut donc des structures plus complexes capables de distinguer les deux types de catégories et d'effectuer des conversions adaptées à chacune.

Opérations sur les matrices de données (paquet `opengis.gp`)

Le calcul effectué par les méthodes `evaluate` peut être modifié. Par exemple, au lieu de retourner la valeur du pixel sous la coordonnée spécifiée, elles pourraient effectuer une interpolation, calculer le gradient en utilisant les pixels environnants, etc. Ces opérations peuvent être appliquées à des données en utilisant la classe `GridCoverageProcessor`. Cette dernière opère en utilisant les informations suivantes :

– L'opération à appliquer. Il peut s'agir d'un nom (par exemple, `GradientMagnitude`). La liste des opérations disponibles apparaît dans la documentation *JAI*.

– Une liste de paramètres (`ParameterList`). Le contenu de cette liste dépend de l'opération à appliquer, mais devrait contenir au moins l'objet `GridCoverage` source.

Le résultat est un nouveau jeu de données `GridCoverage` qui contiendra le résultat de l'opération demandée. Les données originales (c'est-à-dire l'objet `GridCoverage` source) ne sont jamais modifiées. Retenir à la fois les jeux de données originaux et modifiés peut ressembler à un gaspillage de mémoire, mais dans la pratique `GridCoverageProcessor` ne calcule pas immédiatement le résultat de l'opération. Il construit plutôt une chaîne de traitements qui calculera les données à la volée, seulement lorsqu'elles seront nécessaires.

Techniquement, `GridCoverageProcessor` s'appuie sur la bibliothèque *JAI* pour effectuer les opérations. Cette bibliothèque de Sun Microsystems est conçue pour les traitements d'images en général, que ce soit dans le domaine satellitaire, médical ou astronomique. Son lot d'opérations, déjà assez riche, est utilisable avec `GridCoverageProcessor`. Nous ne faisons qu'y ajouter la notion de positionnement géographique propre aux SIG.

Appels à distance avec la technologie RMI (*Remote Method Invocation*)

OpenGIS a conçu ses spécifications dans l'esprit d'une organisation où les données et les calculs seraient distribués sur plusieurs machines distantes. Les technologies utilisées à cet effet (CORBA en C/C++, RMI en Java) placent des contraintes qui peuvent être pénibles lorsque l'on ne travaille que sur un poste local. Nous avons recherché un compromis en ne prenant en compte la technologie RMI qu'au niveau des adaptateurs qui font le lien entre les interfaces standards (`org.opengis`) et nos classes modifiées (`net.seas.opengis`). De cette architecture, on peut déduire les lignes directrices suivantes :

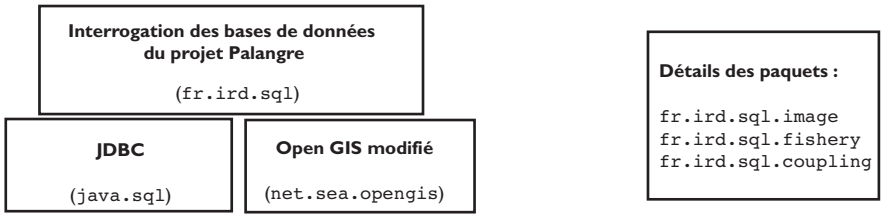
– Pour une utilisation plus aisée en local et une meilleure intégration avec les standards existants de Java, vous pouvez utiliser les classes des paquets `net.seas.opengis`.

– Pour un respect strict des spécifications d'OpenGIS, ou pour une utilisation distribuée à travers un réseau, utilisez les interfaces standards des paquets `org.opengis`.

Ce bref aperçu des spécifications d'OpenGIS sert essentiellement à introduire les notions nécessaires à la compréhension de ce qui suit.

Accès aux bases de données environnementales et de pêches

En utilisant comme brique de base le standard JDBC[11] pour les connections aux bases de données et le standard OpenGIS pour les représentations spatiales, nous avons bâti une interface API facilitant l'exploitation des données environnementales et de pêches (fig. 128). Chaque base de données dispose d'un jeu de classes dans un paquet qui lui est propre (`image` et `fishery` dans le cadre de droite). Un troisième paquet (coupling) fournit quelques utilitaires pour effectuer des croisements entre les deux bases de données.



▽ *Fig 128*

Ces paquets tentent de fournir une interface API de haut niveau qui ne nécessite pas de connaissance du langage SQL. L'utilisateur interroge l'API en spécifiant les coordonnées spatio-temporelles des données qui l'intéressent, l'opération qu'il veut appliquer, etc. L'API construit des requêtes SQL en fonction de ces demandes, procède automatiquement à la lecture des images lorsque c'est nécessaire et construit lui-même les objets OpenGIS correspondants (notamment `GridCoverage`, qui est d'une construction assez laborieuse).

Obtention des données environnementales (paquet `fr.ird.sql.image`)

Toutes les données environnementales sont accessibles par la méthode « evaluate » décrite p . Cette méthode évalue toujours une valeur dans un objet `Coverage` donné. De cet objet dépend le paramètre évalué (température, courants, etc.), l'opération (valeur interpolée, gradient, etc.) ainsi que la date des données. Les classes décrites ici ont pour but ultime de produire des objets `Coverage` en fonction de critères simples et en puisant les informations nécessaires dans la base de données d'images. Ces classes s'enchaînent dans

11. JDBC

une suite d'opérations qui font intervenir plusieurs tables, mais la séquence principale peut être résumée comme suit :

`ImageDataBase` fi ⇒ `ImageTable` ⇒ `ImageEntry` ⇒ `GridCoverage`

La classe `ImageDataBase` représente une connexion à la base de données d'images. Un seul objet de cette classe suffit pour une session de travail. Un objet `ImageDataBase` peut produire un ou plusieurs objets `ImageTable`. Chaque objet `ImageTable` est une vue sur la table **Images** de la base de données (voir plus haut). Cette vue peut être modifiée en fonction de différents paramètres, par exemple les coordonnées spatio-temporelles de la région d'intérêt. Les objets `ImageTable` peuvent eux-mêmes produire un ou plusieurs objets `ImageEntry`, qui représentent chacun une entrée (ou ligne) de la table des images. On peut considérer un objet `ImageEntry` comme une fl vers une image, plutôt que l'image elle-même. Les données de l'image ne sont chargées par `ImageEntry` que sur demande et retournées sous forme d'objets `GridCoverage`. Le tableau 24 illustre les classes que nous venons d'énumérer.

▽ Fig. 24

Image Data Base

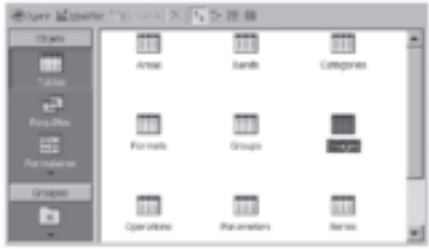


Image Table

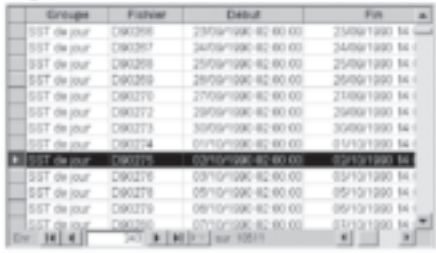


Image Entry et Grid Coverage

Les paragraphes suivants expliquent pas à pas les lignes de code qui doivent être écrites pour accéder à des données à partir d'un programme Java. La première étape est d'obtenir une connexion avec la base de données. Cette étape n'a besoin d'être faite qu'une fois pour toutes au début d'une session de travail.

```
ImageDataBase database = new ImageDataBase ()
```

Les connexions établies de cette façon utilisent la configuration spécifiée plus haut. Si la configuration par défaut ne convient pas, `ImageDataBase` dispose d'un autre constructeur permettant de spécifier explicitement le nom de la base de données, le nom de l'utilisateur et un mot de passe. À partir de cette connexion, on obtient une vue de la table **Images** pour un paramètre donné.

```
ImageTable imageTable = database.getImageTable (SST (synthèse))
```

La ligne précédente construisait une table pour les images de température. Le texte spécifié en argument doit être un des noms qui apparaissent dans la table **Séries** de la base de données. Typiquement, on répètera cette ligne plusieurs fois pour construire plusieurs tables, une pour chaque paramètre d'intérêt. Par défaut, ces tables couvrent toute la plage de coordonnées spatio-temporelles disponibles avec la résolution maximale disponible. Il est possible de se restreindre à une région d'intérêt plus réduite. L'exemple suivant sélectionne une région autour de l'île de la Réunion allant de 50 à 60° E et 15 à 25° S pour la période du 24 au 29 août 1998. Les quatre premières lignes ne font intervenir que des classes de la bibliothèque standard du Java.

```
DateFormat parser = DateFormat.getDateInstance (Locale.FRANCE)
Date startTime = parser.parse (24/08/1998)
Date endTime = parser.parse (29/08/1998)
Rectangle2D area = new Rectangle2D.Double (50, -25, 10, 10)

imageTable.setTimeRange (startTime, endTime)
imageTable.setGeographicArea (area)
```

Si l'on souhaite non pas la valeur du paramètre, mais plutôt le résultat d'une opération (par exemple, le gradient du paramètre), on peut spécifier le nom de l'opération désirée. Cette opération sera transmise automatiquement à `GridCoverageProcessor` le moment venu. L'exemple ci-dessous demande un calcul de gradient en utilisant les opérateurs horizontaux et verticaux de Sobel.

```
imageTable.setOperation (GradientMagnitude)
```

Maintenant que la table est configurée, on peut demander les images qui interceptent les coordonnées spatio-temporelles spécifiées. Deux méthodes existent à cet effet : `getEntries` retourne la liste de toutes les images qui interceptent la région et la plage de temps, tandis que `getEntry` sélectionne une image parmi les candidates. La sélection se fait en privilégiant celle qui couvre le mieux et est la mieux centrée sur la plage de temps spécifiée (les critères exacts sont

détaillés dans la documentation *jdk* , classe `ImageComparator`). L'exemple suivant utilise une une première méthode.

ImageEntry entry = imageTable.**getEntry** ()

Cette méthode est souvent utilisée en couple avec `setTimeRange` et spécifie une courte plage de temps (correspondant par exemple à une pêche), demande la meilleure image correspondante, change la plage de temps, redemande une nouvelle image, etc. Les autres paramètres de `ImageTable` n'ont pas besoin d'être respécifiés à chaque fois s'ils ne changent pas.

`ImageEntry` n'est qu'une référence vers une image. Elle permet de connaître quelques propriétés de l'image (dates exactes, nom de fichier, format, etc.) sans procéder immédiatement à sa lecture. Le chargement des données, s'il est nécessaire, n'est effectué qu'au stade suivant (code ci-dessous). En outre, un système de caches permet à `ImageEntry` d'éviter de faire des lectures redondantes si l'image se trouve déjà en mémoire. Le programmeur n'a donc pas besoin de maintenir un bilan des images qu'il a déjà lues ; c'est fait automatiquement.

GridCoverage coverage = entry.**getGridCoverage** (null)

L'argument (`null` dans cet exemple) permet à ceux qui le désirent, de suivre les progrès d'une éventuelle lecture. Si la lecture a réussi, l'utilisateur peut maintenant extraire les données en utilisant la méthode `evaluate` plus haut. L'exemple ci-dessous utilise une boucle pour afficher les valeurs de la première bande (numérotée 0) à différentes longitudes le long de la latitude 22° 30' S. Le même tableau est réutilisé à chaque passage dans la boucle pour plus d'efficacité, écrasant les anciennes valeurs. Par défaut, les valeurs sont interpolées en utilisant une interpolation bi-cubique[12].

```
float[] values = null
Point2D coord = new Point2D.Double ()[13]
for (double x=55.0 ; x<=60.0; x+=0.5)
{
    coord.setLocation (x, -22.5)
values = coverage.evaluate (coord, values)
System.out.println (values[0])
}
```

Interpolations des données environnementales dans le temps

La méthode décrite dans la section précédente permettait d'obtenir les données environnementales correspondant à une image. Une interpolation était faite dans l'espace, mais pas dans le temps. Cette section introduit une deuxième méthode plus simple qui ajoutera en plus cette interpolation dans le temps.

12. Si l'interpolation bi-cubique échoue parce qu'un nuage masquait des pixels environnants, la méthode se rabat sur une interpolation bi-linéaire. Si cette dernière échoue aussi, alors cette méthode retourne la valeur du plus proche voisin.

13. Dans cet exemple, nous ignorons la troisième dimension (le temps) puisque nous travaillons sur une seule image. Nous pouvons utiliser dans ce cas-ci un objet Point2D plutôt qu'un objet CoordinatePoint plus général.

Le début de la procédure est le même, il faut configurer un objet `ImageTable` avec le paramètre, l'opération et les coordonnées spatio-temporelles désirés. Par la suite, au lieu d'extraire des images individuelles avec les méthodes `getEntry` ou `getEntries`, on peut confier plutôt cette tâche à un objet `Coverage3D`, qui nous donnera une vue à trois dimensions des données.

Coverage3D coverage3D = new **Coverage3D** (imageTable)

`Coverage3D` est une extension de la classe `Coverage` d'OpenGIS, adaptée à l'API décrit plus haut. À partir du moment où il est construit, on peut oublier la table et fermer complètement la connexion avec la base de données. L'extraction des valeurs se fait en spécifiant les coordonnées spatio-temporelles désirées, sans se soucier de savoir sur quelles images elles se trouvent. Les valeurs sont interpolées à la fois dans l'espace et dans le temps, toujours à la condition qu'il n'y ait pas de trous dans les données. L'exemple ci-dessous interpole la donnée du 27 août 1998 à la coordonnée 55° E, 20° S. Nous réutilisons les mêmes objets `parser` et `values` que dans les exemples précédents.

```
Date time = parser.parse (27/08/1998)
Point2D point = new Point2D.Double (55,-20)
values = coverage3D.evaluate (point, time, values)
```

Obtentions des données de pêches (paquet `fr.ird.sql.fishery`)

Les données de pêches s'obtiennent d'une façon tout à fait similaire aux trois premières étapes de l'obtention des données environnementales. Il faut d'abord obtenir une connexion avec la base de données des captures (indépendamment de la base de données d'images), une table des captures et enfin les données des captures. Les classes impliquées s'enchaînent comme suit :

`CatchDataBase` ⇒ `CatchTable` ⇒ `CatchEntry`

Comme pour les images, la première étape est d'obtenir une connexion avec la base de données. Cette étape n'a besoin d'être faite qu'une fois pour toutes au début d'une session de travail.

CatchDataBase database = new **CatchDataBase** ()

D'autres constructeurs sont disponibles si l'on souhaite spécifier un nom d'utilisateur et un mot de passe. L'étape suivante est d'utiliser cette connexion pour obtenir une vue de la table **Captures**.

CatchTable catchTable = database.**getCatchTable** (Captures)

Le texte spécifié en argument donne le nom de la table à examiner. Pour le projet Palangre, il n'y a qu'une seule table des captures. D'autres projets pourraient toutefois utiliser plusieurs tables distinctes, par exemple une table pour les captures à la palangre et une autre pour les captures à la senne. Par défaut, ces tables couvrent toute la plage de coordonnées spatio-temporelles disponibles. Il est possible de se restreindre à une région d'intérêt plus réduite en procédant exactement de la même façon que pour les images. L'exemple suivant sélectionne une plage de temps allant du 24 au 29 août 1998.

```
DateFormat parser = DateFormat.getDateInstance (Locale.FRANCE) ;
Date startTime = parser.parse (24/08/1998)
Date endTime = parser.parse (29/08/1998)
catchTable.setTimeRange (startTime, endTime)
```

Une fois la table configurée, on peut demander les captures qui interceptent les coordonnées spatio-temporelles spécifiées. La méthode `getEntries` retourne la liste de toutes les captures qui répondent aux conditions, comme dans l'exemple suivant :

```
List catchs = catchTable.getEntries ()
```

La liste ne contient que des objets `CatchEntry`. Ces derniers représentent des captures à une certaine date dans une région géographique. Dans le cas des données palangrières, chaque objet `CatchEntry` correspond à une ligne de palangre mise à l'eau et récupérée. Ces objets contiennent entre autres les fonctions suivantes :

`getCoordinate ()`

Retourne une coordonnée géographique représentative de la capture. Pour les palangres, c'est généralement la coordonnée du milieu de la ligne.

`getShape ()`

Retourne une forme géométrique représentative de la zone de pêche. Pour les palangres, c'est généralement un objet `Line2D` représentant une ligne droite reliant la position de début à la position de fin de la ligne. Cette forme n'est qu'approximative et peut être plus complexe qu'une simple ligne si davantage d'informations sont disponibles.

`getTime ()`

Retourne une date représentative de la capture. Dans le cas des données du projet Palangre, l'heure de l'objet `Date` retourné n'est pas significative.

`getCatch (Species)`

Retourne les captures pour une espèce donnée (spécifiée en argument). La méthode `getUnit` () doit être interrogée pour connaître les unités. Il pourrait s'agir du nombre d'individus, du tonnage ou de n'importe quelles autres unités valides. Les données du projet Palangre utilisent toujours le nombre d'individus par 1 000 hameçons.

Croisement entre les données environnementales et halieutiques

Les API décrites dans les sections précédentes offrent déjà les fonctionnalités nécessaires au croisement des données. À partir d'une donnée de pêche, on peut facilement obtenir les données environnementales à la position correspondante. Le croisement des données étant un aspect essentiel du projet Palangre, quelques classes supplémentaires sont tout de même dédiées exclusivement à cette tâche. Ces classes supplémentaires sont regroupées dans le paquet `fr.ird.sql.coupling`. Ce dernier combine les usages des deux paquets indépendants introduits dans les sections précédentes (`fr.ird.sql.image` et `fr.ird.sql.fishery`).

`CatchCoverage`

La classe `CatchCoverage` hérite de la classe `Coverage3D` introduite plus haut. Elle y ajoute une méthode `evaluate` qui reçoit en argument une donnée de capture `CatchEntry`. Cette méthode détermine elle-même la date et les coordonnées géographiques des données environnementales à rechercher, en fonction de la donnée de capture spécifiée. Contrairement aux autres méthodes `evaluate`, celle de `CatchCoverage` est capable d'évaluer les paramètres environnementaux dans une région géographique autour de la capture plutôt qu'en un seul point. Elle peut calculer par exemple la température moyenne ou le gradient maximal trouvé dans un rayon de 10 milles nautiques autour de la capture. La forme de la région géographique peut dépendre de la capture. Par exemple, dans le cas d'une ligne de palangre, la région géographique pourrait être une ellipse dont les foyers coïncident avec le début et la fin de la ligne de pêche. Une telle ellipse peut être un compromis tenant compte du fait que l'on ne connaît pas exactement la trajectoire de la ligne entre ces deux points.

`CatchTableFiller`

La classe `CatchTableFiller` examine les coordonnées de début et de fin des lignes de palangres pour calculer la distance orthodromique entre ces deux points, la distance minimale d'avec les autres lignes du même jour et la distance minimale de la côte. Ces informations sont enregistrées dans les colonnes correspondantes de la table **Captures** de la base de données des pêches.

`EnvironmentTableFiller`

La classe `EnvironmentTableFiller` extrait les données environnementales aux positions de pêches 15, 10 et 5 jours avant le jour de la pêche et 5 jours après. Les informations trouvées sont enregistrées dans les colonnes correspondantes de la table **Environnements** de la base de données des pêches.

Les classes `CatchTableFiller` et `EnvironmentTableFiller` sont toutes deux des applications. C'est-à-dire qu'elles sont tout en haut d'une chaîne de traitement et qu'elles ont une méthode `main` qui permet de les lancer à partir d'une ligne de commande. Ces deux classes sont relativement simples, puisque la complexité du travail est distribuée dans toutes les autres classes décrites depuis le début. En conséquence, les développeurs qui souhaiteraient effectuer des calculs différents sont invités à modifier le code de ces deux classes `TableFiller`. Cette approche laisse une liberté considérablement plus grande que tout ce que permettraient des boîtes de dialogues.

Enregistrements des activités dans un journal

Les méthodes décrites plus haut génèrent des événements qui peuvent être enregistrés dans un journal. Ce journal remplit deux fonctions. Il sert d'abord d'accusé de réception pendant le calcul, vous confirmant que chacune des instructions décrites dans les chapitres précédents a bien été exécutée. Il sert aussi d'archive pour le calcul, au cas où l'on voudrait vérifier les opérations qui ont été effectuées. Les journaux sont gérés et documentés par le Java standard.

Les événements se déclinent en plusieurs niveaux. En allant du plus « sévère » (et rare) au plus détaillé (et abondant), on compte les niveaux `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER` et `FINEST`. Les méthodes qui modifient l'état d'une table (par exemple, pour spécifier des coordonnées spatio-temporelles) génèrent des événements de niveau `CONFIG`. Les méthodes qui extraient des données d'une table génèrent des événements de niveau `FINE` (ces événements étant généralement plus abondants que la configuration d'une table, ils doivent être d'un niveau inférieur).

Par défaut, le Java enregistre les événements dans un fichier XML et en filtre une partie qu'il envoie à l'écran. Seuls les événements de niveau INFO ou supérieur apparaissent à l'écran. Cette configuration par défaut peut être modifiée en éditant le fichier `jdk1.4/jre/lib/logging.properties` (le chemin exact dépend du système). Pour autoriser l'affichage des événements décrits dans les sections précédentes, il faut ajouter ou modifier les lignes sur- et soulignées ci-dessous.

```
–
# H[illegible]
# D[illegible]H[illegible]
–
# [illegible]                                    FINE [illegible]
java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter


–
# F[illegible]
# [illegible]
–
fr.ird.level = FINE
```

Affichages graphiques des données

Certains objets peuvent être représentés graphiquement à l'écran. Les capacités d'interactions avec l'utilisateur sont limitées, puisqu'il ne s'agit pas de dupliquer ici les fonctionnalités des logiciels commerciaux.

Afficher graphiquement la liste des images et séries disponibles

On peut afficher le contenu d'une table `ImageTable` en l'enveloppant dans une composante graphique [illegible]. L'exemple ci-dessous configure une composante [illegible] `JTable` à partir des données de la table d'images utilisée dans les exemples précédents. Cette vue de la table représentera en bleu les images qui ont déjà été lues et en rouge les images qui n'ont pas été trouvées. Cette coloration est dynamique dans le sens que si une des images est lue (même à travers un réseau), la ligne correspondante de la table prendra automatiquement la couleur bleue sans que le programmeur n'ait à s'en soucier.

```
TableCellRenderer renderer = new ImageTableModel.CellRenderer ()
JTable tableView = new Jtable (new ImageTableModel (imageTable))
tableView.setDefaultRenderer (String.class, renderer)
tableView.setDefaultRenderer (Date.class, renderer)
```

Nous ne détaillons pas l'API utilisé, car il n'est pas de notre ressort d'expliquer l'architecture de [illegible]. Il existe déjà de nombreux ouvrages à cette fin. Le prochain exemple construit l'arborescence des séries disponibles dans une composante [illegible] `JTree`.

```
SeriesTable seriesTable = database.getSeriesTable ()
TreeModel tree = seriesTable.getTree (SeriesTable.SERIES_LEAF)
JTree treeView = new Jtree (tree)
```

AFFICHER GRAPHIQUEMENT UNE IMAGE ET LES PÊCHES CORRESPONDANTES

Le principe utilisé est le même que celui des logiciels commerciaux : on commence avec une fenêtre vide, qui agit comme un réceptacle sur lequel on superpose une ou plusieurs couches. La fenêtre vide est une composante [illegible] que nous avons créée : `MapPanel`. Cette composante prend en charge la gestion des zooms et connaît le système de coordonnées de l'affichage, qui n'est pas nécessairement le même que celui des données.

Les couches sont de plusieurs types. Elles sont toutes des classes dérivées de `Layer`, mais la classe exacte dépendra du type d'information à faire apparaître. Il peut s'agir de points ou de lignes indiquant les positions des pêches, des flèches de courant, des ellipses de marée, des images satellitaires, des courbes de pression, l'échelle de la carte, des frontières géopolitiques, des villes, etc. Comme cette bibliothèque ne peut pas embrasser toutes les étendues des possibilités, les quelques classes qu'elle propose sont conçues de façon à faciliter autant que possible la tâche du programmeur qui voudrait les étendre.

Le fonctionnement de ce paquet présente quelques similitudes avec les composantes AWT du Java standard, qui servent à construire une interface utilisateur. Pour mieux comprendre le fonctionnement de ce paquet, il est utile de faire un parallèle entre celui-ci et AWT. Cette documentation suppose donc que le lecteur est familiarisé avec le paquet `java.awt`. Une attention toute particulière devrait être apportée aux systèmes de coordonnées des objets `Graphics2D`.

Rappel de quelques notions de AWT

Une application fenêtrée contient une liste de composantes formant l'interface utilisateur. Chaque type de composante (bouton, barre de défilement, etc.) est représenté par une classe spécialisée (`Button`, `Scrollbar`, etc.) qui dérive obligatoirement de la classe `Component`. Quand le système a déterminé qu'il faut redessiner une composante, il appelle automatiquement la méthode `paint(…)` de celle-ci. Lors de l'appel, il transmet à chaque méthode `paint(…)` un même objet `Graphics2D` temporaire qu'il aura préalablement créé et configuré selon le tableau 25 :

▽ [illegible] 25

[illegible] Graphics2D [illegible] AWT

Origine (0,0) des axes	Coin supérieur gauche de l'écran ou du papier d'imprimante
Direction des axes	Les *x* croissent vers la droite et les *y* vers le bas
Unités des axes	Les pixels (sorties à l'écran) ou environ 1/72 de pouce (sorties à l'imprimante)
Épaisseur des lignes	Un pixel (sorties à l'écran) ou environ 1/72 de pouce (sorties à l'imprimante)

Cette configuration convient très bien à l'écriture de texte. Ainsi, un texte de 12 unités de haut aura une hauteur de 12 pixels à l'écran ou de 12/72 de pouces à l'impression. Les méthodes `paint(…)` peuvent changer temporairement la configuration de `Graphics2D` lors d'un traçage, mais doivent le remettre dans son état initial lorsqu'elles se terminent.

Un objet `MapPanel` peut contenir une liste de couches représentant les informations que l'on place sur une carte. Chaque type de couche (stations, échelle de la carte, etc.) est représentée par une classe spécialisée (`MarkLayer`, `MapScaleLayer`, etc.) qui dérive obligatoirement de la classe `Layer`. Quand le système a déterminé qu'il faut redessiner une couche, il appelle automatiquement la méthode `paint (…)` en transmettant à chacun un même objet `Graphics2D` temporaire qu'il aura préalablement créé et configuré selon le tableau 26 :

Tableau 26

Configuration de Graphics2D lors du traçage de couches cartographiques

Origine (0,0) des axes	Dépend de la projection cartographique en cours Souvent en dehors de la région visible de la carte
Direction des axes	Les *x* croissent vers la droite et les *y* vers le haut, comme en géométrie
Unités des axes	Toujours en mètres sur le terrain (et non en mètres sur l'écran !)
Épaisseur des lignes	Dépend de la résolution de la carte Peut être de l'ordre de 50 mètres

Cette configuration convient très bien au traçage de couches cartographiques. Elle permet de travailler avec les dimensions réelles des constructions sans se soucier du facteur d'échelle. Toutefois, elle ne convient pas du tout à l'écriture de texte. Par exemple, un texte de 12 unités de haut sera interprété comme ayant une hauteur de 12 mètres. Pour une carte à l'échelle 1/50 000, des lettres de 12 mètres apparaîtront à l'écran comme de minuscules points. Le programmeur peut régler le problème en changeant temporairement la configuration de `Graphics2D`, mais doivent le remettre dans son état initial lorsqu'il aura terminé.

Rappel de quelques notions sur les transformations affines

Pour convertir en pixels des coordonnées exprimées selon un autre système, Java2D utilise une transformation affine représentée par la classe `AffineTransform`. En résumé, une transformation affine est une matrice 3 x 3. En plaçant dans cette matrice les bons coefficients, on peut obtenir n'importe quelle combinaison d'échelles, translations, rotations et cisaillements. Dans notre cas particulier, nous utilisons cette matrice de transformation affine pour convertir en pixels des coordonnées exprimées en mètres :

$$\begin{bmatrix} x_{\text{pixels}} \\ y_{\text{pixels}} \\ 1 \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_{\text{mètres}} \\ y_{\text{mètres}} \\ 1 \end{bmatrix}$$

La classe `AffineTransform` fournit un ensemble de méthodes permettant d'appliquer des transformations sur cette matrice sans nécessairement connaître les mathématiques sous-jacentes. Par exemple, la méthode `rotate(…)` modifiera les termes de la matrice 3 x 3 pour y ajouter l'effet d'une rotation d'un angle quelconque. Notons au passage une identité qui se révèle parfois pratique :

$$\sqrt{(m_{00})^2 + (m_{01})^2} \quad \text{et} \quad \sqrt{(m_{10})^2 + (m_{11})^2}$$

sont invariants sous rotations

En l'absence de rotation ou de cisaillement, les termes m_{01} et m_{10} sont nuls. Restent alors les termes m_{00} et m_{11}, qui nous indiquent l'échelle de la carte telle qu'appliquée par la transformation affine. Dans cette situation simplifiée on obtient donc les relations suivantes :

$\sqrt{(m_{00})^2 + (m_{01})^2}$ m_{00} = `AffineTransform.getScaleX ()`

$\sqrt{(m_{10})^2 + (m_{11})^2}$ m_{11} = `AffineTransform.getScaleY ()`

Utilisation par interface graphique de SEASview pour le non-développeur

Principe du logiciel

Le logiciel de visualisation vise à faciliter les accès à une base de données comprenant l'ensemble des données satellitaires fournies par l'IRD et des données d'abondance fournies par l'Ifremer. Les cartes sont le fruit de mesures satellitaires et de calculs expliqués dans les chapitres précédents. Elles contiennent de plus une information géographique. Le déplacement du curseur fournit à l'utilisateur une position géographique dans un référentiel déterminé. Le référentiel choisi est l'ellipsoïde WGS 84. Ce référentiel déforme non seulement moins que d'autres référentiels une zone aussi étendue que l'océan Indien, mais il est aussi très largement utilisé en particulier par le système de localisation GPS[14]. Grâce à cette information géographique, les prises des pêcheurs fournies par l'Ifremer pourront être positionnées.

L'utilisateur pourra en conséquence observer le paysage sur quatre cartes indiquant quatre paramètres océaniques différents lors des prises des pêcheurs, et ce à une date et sur une zone choisie.

Pour utiliser le logiciel, le contenu de tous les cédéroms peut être copié sur le disque dur du logiciel, mais alors la place requise sera importante. Sinon, et cette option sera recommandée, il suffit de procéder à l'installation du cédérom #1 contenant les applications, comme indiqué ci-après ; le logiciel SEASview précisera les cédéroms à insérer pour chaque visualisation.

14. *GPS*

Installation du logiciel

La programmation a été réalisée en langage Java. Ce langage a la particularité de pouvoir être utilisé quel que soit le système d'exploitation (Unix, VAX-VMS, Linux, Windows…). Néanmoins, il demande l'installation d'une interface pour que chaque système puisse exécuter les programmes. Plusieurs utilitaires conditionnent l'installation de SEASview. Chaque étape doit être suivie rigoureusement pour une exécution correcte du logiciel.

Java Run-time Environment (JRE 1.3.0)

SEASview nécessite la version 1.3.0 ou ultérieure de l'environnement d'exécution Java. Si vous disposez d'une connexion rapide à Internet, nous vous recommandons de télécharger la version la plus récente directement du site de Sun Microsystems :

http://java.sun.com/j2se/1.3/jre/

Si vous ne souhaitez pas effectuer ce téléchargement, vous pouvez utiliser le programme d'installation fourni sur le cédérom #1. L'installation s'effectue en exécutant le programme suivant (pour les utilisateurs de Windows) :

Install/Windows/j2re-1_3_0_01-win-i.exe

Il existe aussi des répertoires Linux et Solaris pour les utilisateurs de ces systèmes d'exploitation.

Comment vérifier si le Java est déjà installé sur mon système ?

Tapez l'instruction suivante sur la ligne de commande :

java–version

Vous devriez obtenir un affichage ressemblant à :

java version 1.3.0
Java (TM) 2 Runtime Environment, Standard Edition (build 1.3.0-C)
Java HotSpot (TM) Client VM (build 1.3.0-C, mixed mode)

L'affichage peut varier. L'exemple vaut pour le Java de SUN Microsystems. Le Java d'IBM donnerait un affichage légèrement différent, mais qui serait tout aussi bon, s'il indique le bon numéro de version (sur la première ligne). Si les lignes qui apparaissent indiquent la version 1.3.0 de Java (ou une version plus récente), il est inutile de réaliser l'installation décrite dans cette section. Sautez directement à l'étape 3.2.2.

Java Advanced Imaging (JAI 1.1)

En plus de l'environnement standard, SEASview nécessite aussi une bibliothèque de traitement d'images. Vous pouvez l'installer à partir du cédérom #1 en exécutant le programme suivant :

Install/Windows/jai-1_1-beta-lib-win-jre.exe

Installation de diverses extensions

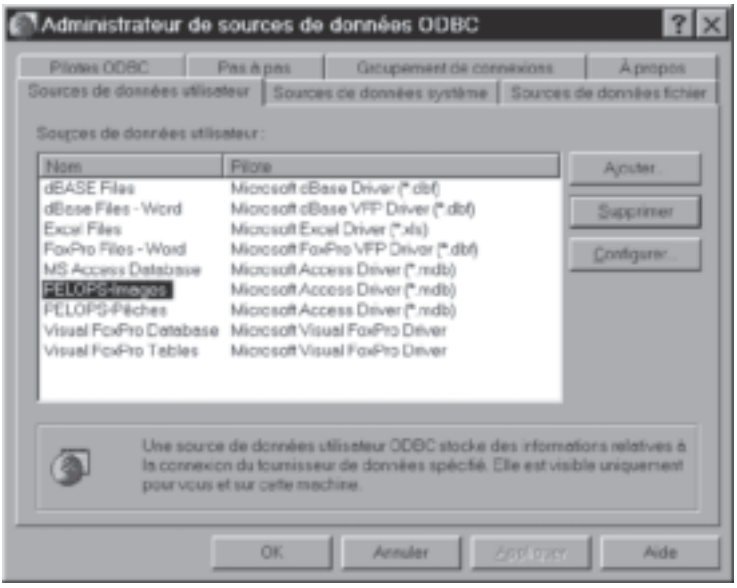
Copiez les quatre fichiers du dossier `Instal/Windows/ext` dans `Program Files/Javasoft/JRE/1.3.0_01/lib/ext`.

LES BASES DE DONNÉES

Les bases de données sont au nombre de deux : la BD/Images et la BD/Pêche. Elles ont été construites sous Access, mais l'installation du logiciel Access n'est pas indispensable pour l'exploitation de la base avec le logiciel de visualisation.

1) Copiez les bases de données d'images et de pêches sur votre disque dur. Elles peuvent être placées dans n'importe quel répertoire (par exemple, dans un dossier `//Palangre/Seasview`).

2) Déclarez les bases de données au système, en procédant comme suit : allez dans le panneau de configuration (bouton Démarrer/Paramètres/Panneau de configuration) et ouvrez l'icône **Source de données ODBC 32-bits**. Une fenêtre similaire à la fenêtre1 devrait apparaître.



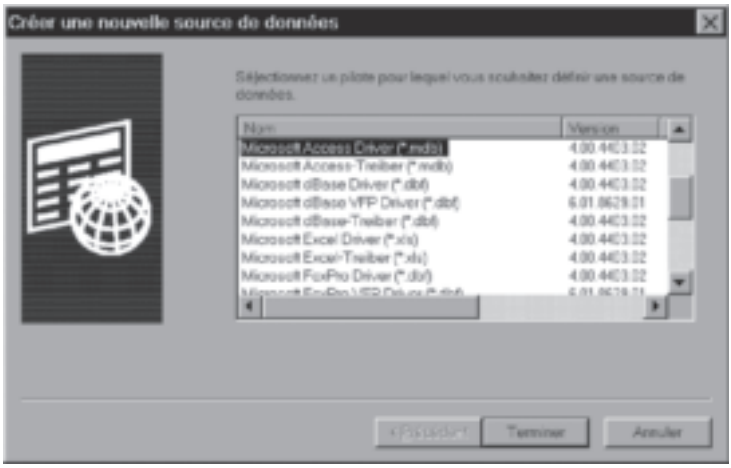
▽ *Fig.*

Appuyez sur **Ajouter**. Une fenêtre similaire à la fenêtre 2 devrait apparaître.

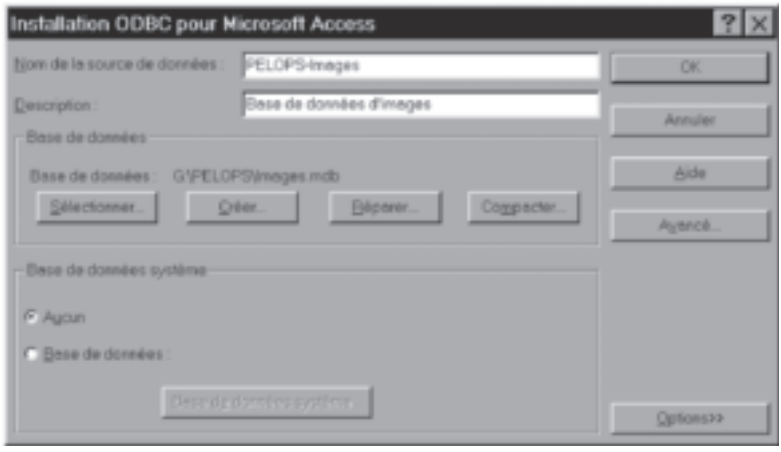
Sélectionnez **Microsoft Access Driver** et cliquez sur **Terminer**. La fenêtre 3 devrait alors apparaître.

Dans le champ **Nom de la source de données**, il faut entrer **SEAS-Images** (en respectant les majuscules et les minuscules). Le contenu du champ Description est laissé libre, vous pouvez y entrer les commentaires de votre choix. Cliquez ensuite sur Sélectionner et sélectionnez le fichier **Images.mdb** (sa position sur le disque dur dépend de l'endroit où vous l'avez mis). Cliquez ensuite sur Ok et fermez l'autre fenêtre en cliquant aussi sur Ok.

Pour la deuxième base de données, refaites les mêmes opérations que précédemment,. Le nom de la source de données est **SEAS-Pêches**, toujours en respectant la typographie, le fichier correspondant **Pêches.mdb**.



▽ Fig. 2.



▽ Fig. 3.

Installation du logiciel

Copiez le fichier SEAS.jar (`Install/Windows`) sur le disque dur, pourquoi pas dans `//Palangre/Seasview` avec les bases de données d'images et de données de pêche.

Il ne reste plus qu'à l'exécuter en double-cliquant sur SEAS.jar.

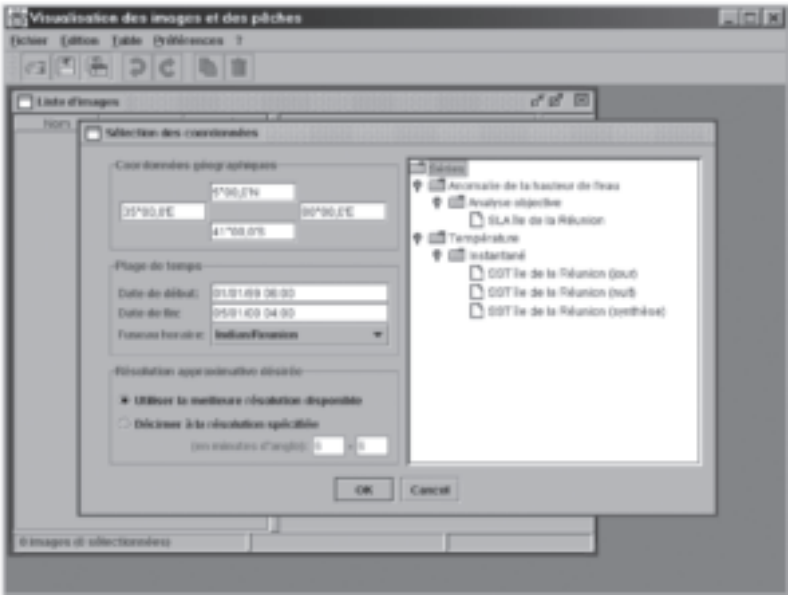
Pour finaliser l'installation du logiciel, il convient de préciser l'emplacement des images. Dans la barre d'outils de l'application, choisissez **Préférences/Base de données** et indiquez, dans le champ **Répertoire racine**, la localisation du répertoire sur le disque dur. Si l'option proposée au début du chapitre a été retenue, il suffit d'indiquer la lettre du lecteur de disque compact du PC. Pour que cette information soit enregistrée, il est impératif de créer un répertoire `/ApplicationData` dans le même répertoire que **SEAS.jar**.

Utilisation du logiciel

Deux modes de visualisation peuvent être utilisés ici : un mode simple et un mode synchronisé. Le mode simple permet de sélectionner un type de données (SST, SLA, Chl-*a*, vorticité) et de voir l'évolution dans le temps du même paramètre. Le mode synchronisé permet de visualiser les paramètres disponibles en un même lieu et à une même date.

LE MODE SIMPLE

La première étape consiste à sélectionner les références des images sur une zone et pour une période donnée. Dans la barre des menus, sélectionnez **Table/Nouvelle table d'images**. La fenêtre 4 apparaît.



▽ *Fig. 4.*

Les paramètres d'entrée sont :

– la zone géographique,
– la plage de temps,
– le fuseau horaire,
– une résolution approximative des images (optionnel),
– les types de données. Les séries sont sélectionnées en cliquant sur l'intitulé.

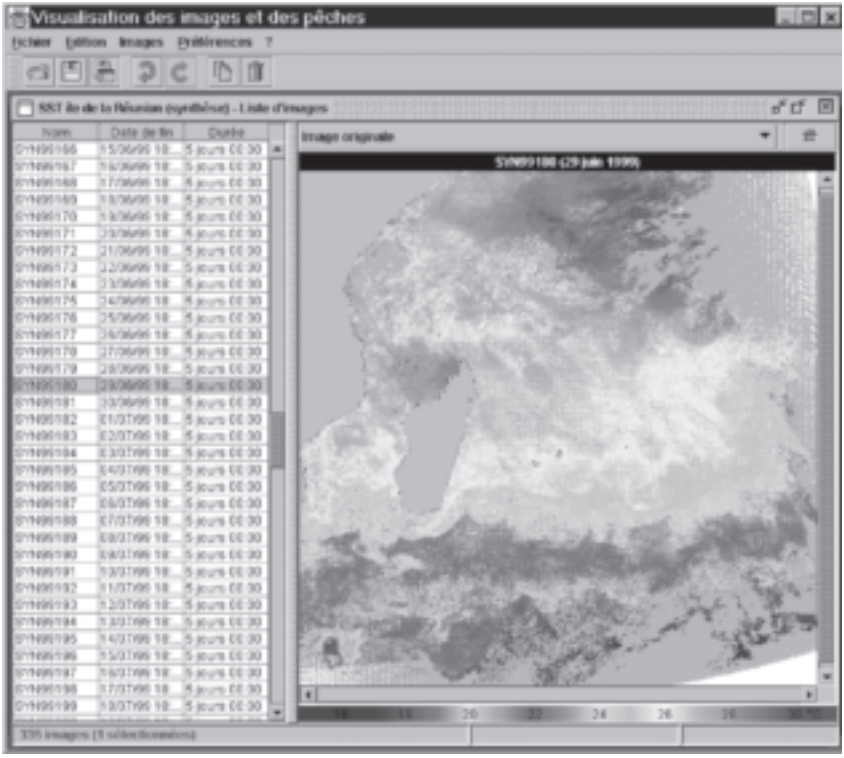
Puis cliquez sur Ok .

Une résolution approximative peut être spécifiée en minutes d'angle. Les fichiers de synthèse des températures de surface sont assez volumineux (au moins 3 Mo), et donc cette option est utile afin de limiter le temps d'affichage lors de premières observations.

Si la résolution choisie est inférieure à 6 minutes d'angle, le logiciel prendra l'image de départ et sous-échantillonnera l'image en prenant un pixel sur 3 (horizontalement et verticalement) si la résolution choisie est 3 minutes d'angle, par exemple.

Si la résolution choisie est supérieure à 6 minutes d'angle, elle s'arrondira au multiple de 6 le plus proche. Le même processus de décimation sera choisi, mais à partir d'images déjà calculées avec une résolution de 6 minutes d'angle et accessibles dans la base de données.

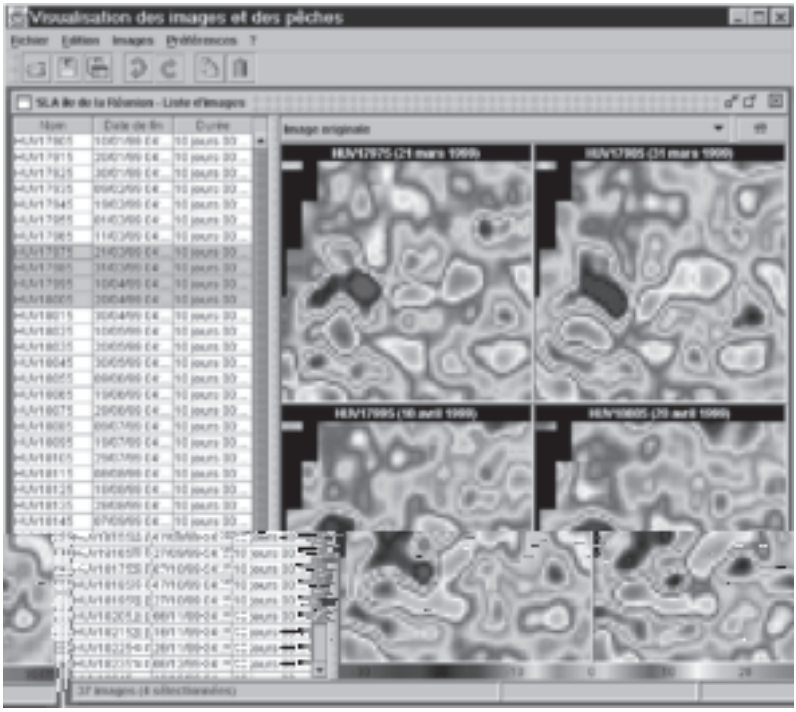
Le choix de la table d'images étant fait, une fenêtre apparaît. À gauche, les noms des images sont affichés et si l'utilisateur clique dessus, l'image apparaît (fig. 129).



▽ *Fig 129*

En bas de l'image apparaît la légende de la carte. En déplaçant le curseur, les coordonnées géographiques du point changent ainsi que la valeur du paramètre.

L'utilisateur peut aussi sélectionner plusieurs images en même temps : en faisant glisser la souris dans la table d'images ou bien en utilisant la touche Ctrl (fig. 130).



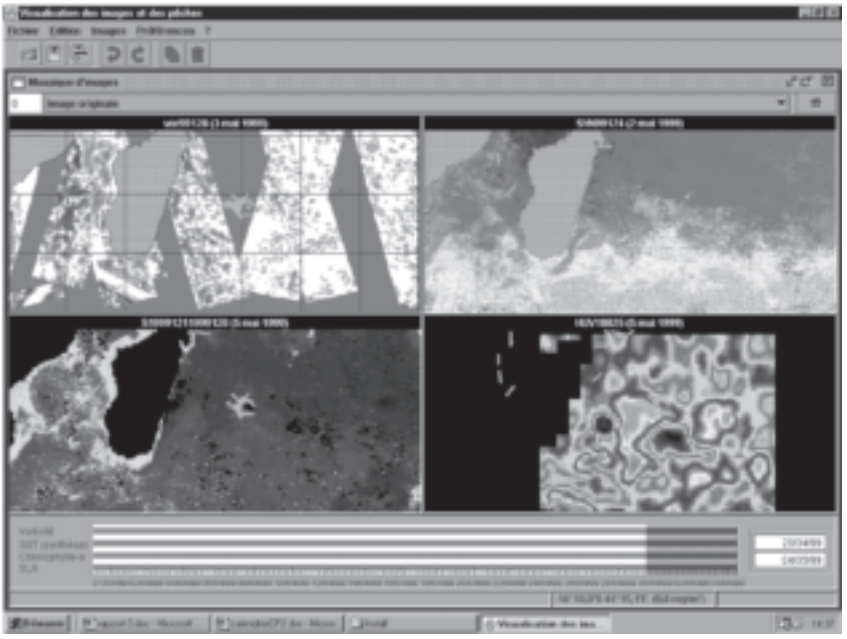
▽ *Fig 130*

▽ *Fig 131*

Un certain nombre d'outils sont accessibles par le bouton droit de la souris sur les images (fig. 131) : pour déplacer l'image dans la fenêtre (haut, bas, rotation...), pour faire un agrandissement de l'image (zoom avant, zoom arrière).

Le bouton juste au-dessus de l'image à droite permet d'ajuster l'image à la fenêtre. Enfin, la barre en haut de l'image permet d'accéder à des options pour l'affichage

SEASview permet la visualisation des données de pêche sur les différentes cartes thématiques () et pourra extraire les mesures croisées avec les données de pêche des différents paramètres. Cette matrice pourra ensuite être analysée par des outils de statistiques classiques (fig. 132).



▽ *Fg132*