

SGBD : développez d'abord, choisissez après !

Entrez ! Soyez le bienvenu dans la maison du développement universel !
N'ayez pas peur ! Je sais, c'est l'environnement qui vous effraie... Il est pourtant si pratique.
C'est du XML partout ! Tables, fenêtres, tout est XML !

Un peu surprenant ?

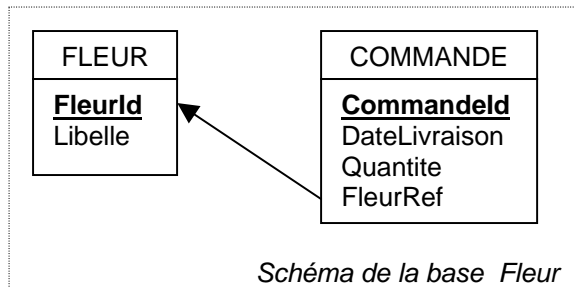
Ce que j'ai fait de la programmation ? Ne vous inquiétez pas pour elle : elle sera toujours là mais vous ne la verrez pas. Peut-être ne le saviez vous pas, le programmation n'a jamais aimé les projecteurs. Pour elle, « projecteurs » signifie « erreurs ». Elle avait déjà disparu du stockage de données en se cachant derrière la norme SQL. Mais côté interfaçage avec l'utilisateur, jamais elle n'avait réussi ! Manque d'imagination peut-être...

Présentation

La technologie Sashipa-Melba permet de construire des interfaces de bases de données sans programmer, juste en décrivant dans un fichier XML au format Sashipa. Pour me suivre, je vous demanderai une connaissance minimale d'un des SGBD/R supportés, rien de plus. Mais assez de causerie, passons à la pratique !

Préliminaire : Création de la base de données

Je vous propose de construire une application de gestion des commandes de fleurs, pour un fleuriste. Je fais donc appel à votre expertise pour créer la base de données.



Sashipa-Melba supporte les SGBD PostGreSQL, Interbase, MySQL, Oracle, SQL Server, Access, HSQL. Choisissez en un parmi ceux-là. Voici le script SQL pour MySQL. (en fin du fichier fleur.xml pour les autres)

```
CREATE TABLE FLEUR (  
    FleurId integer not null primary key,  
    Libelle varchar(100) not null UNIQUE  
);  
CREATE TABLE COMMANDE (  
    CommandeId integer not null primary key,  
    DateLivraison date not null,  
    Quantite integer,  
    FleurRef integer not null REFERENCES FLEUR(FleurId),  
    UNIQUE (DateLivraison, FleurRef)  
);
```

Script de la base Fleur pour MySQL

Apprentissage de Sashipa

Ok maintenant c'est parti.

Le code source de votre application sera constitué d'un unique fichier XML au format Sashipa. De façon automatique, ce fichier sera ensuite transformé en un source Java, puis compilé pour vous donner une application Java.

Le fichier « *fleur.xml* » est un fichier texte classique portant l'extension '.xml'. Vous travaillerez avec votre éditeur de texte préféré (JExt, UltraEdit, Emacs, Notepad, ...).

Ok maintenant c'est vraiment parti !

Un fichier au format Sashipa est composé d'un élément 'application' contenant trois parties principales :

- **L'environnement**, qui décrit votre base de données physique.
- **La GUI**, c'est à dire votre application proprement dite. La description de la GUI s'appuie sur l'environnement.
- **L'Architecture**. C'est ici que sont décrites l'architecture à générer ainsi que les informations techniques de connexions.

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<!DOCTYPE application SYSTEM 'sashipa.dtd'>

<application name='AppliFleur'>
  <environment>
    <!-- ... -->
  </environment>
  <graphicalUserInterface name='guiFleur'>
    <!-- ... -->
  </graphicalUserInterface>
  <architecture>
    <!-- ... -->
  </architecture>
</application>
```

Squelette du source du logiciel Fleur

Première partie : Environnement.

L'environnement décrit la structure de votre base de données, telle qu'elle est stockée dans votre SGBD.

Le contenu est assez simple, je donne ici la description de la table FLEUR. Vous trouverez le code source complet dans le fichier fleur.xml.

On remarque le nom de la table interne au fichier XML : **tblfle**. Ce nom est distinct du nom réel de la table (**FLEUR**). Lorsque d'autres éléments référenceront la table FLEUR, ils utiliseront le nom interne bien sûr. Donc si un jour vous deviez changer physiquement le nom de votre table, il n'y aurait qu'un seul endroit à mettre à jour dans votre source XML. Cette distinction **nom interne / nom physique** est valable pour tous les éléments.

```
<schemaTable name='tblfle'>
  <physicalName>FLEUR</physicalName>
  <singularName>Fleur</singularName>
  <pluralName>Fleur(s)</pluralName>
  <schemaColumnSet>
    <schemaColumn name='tblfle_FleurId' type='integer'
      notNull='yes' pk='yes'>
      <physicalName>FleurId</physicalName>
      <singularName>Identifiant</singularName>
    </schemaColumn>
    <schemaColumn name='tblfle_Libelle' type='text'
      notNull='yes' maxCharacters='100'
      defaultListLetterCount='20'>
      <physicalName>Libelle</physicalName>
      <singularName>Type de Fleur</singularName>
    </schemaColumn>
  </schemaColumnSet>
  <userKey>
    <userKeyColumn schemaColumn='tblfle_Libelle' />
  </userKey>
</schemaTable>
```

Environnement : La table FLEUR

Dernière partie : Architecture.

Avant de voir la GUI proprement dite, jetons un coup d'œil sur la partie Architecture.

On définit en premier le type de logiciel généré. On peut choisir une applet ou une application, comme c'est le cas ici.

C'est ici aussi que sont décrites les connexions vers les bases de données. Les informations sont différentes selon le SGBD que vous choisissez, et le mode de connexion. Je donne ici l'exemple pour MySQL en passant par ODBC. Pour les autres SGBD, il suffit d'intervertir les commentaires dans le fichier fleur.xml.

```
<architecture guiType='application'>
  <dbAccessStage database='dbFleur'
    architecture='clientDatabase'>
    <!-- Exemple pour MySQL -->
    <dbConnection type='odbc' dbmsType='MySQL'>
      <dbConnectionString>
        DRIVER=MySQL;HOST=localhost;DB=Fleur
      </dbConnectionString>
      <user>root</user>
      <password></password>
    </dbConnection>

    <dbAccessLog>
      <filenameLogMain>Melba_Main.log
      </filenameLogMain>
      <filenameLogUpdate>Melba_Update.log
      </filenameLogUpdate>
      <filenameLogSelect>Melba_Select.log
      </filenameLogSelect>
    </dbAccessLog>
  </dbAccessStage>
</architecture>
```

La couche d'accès aux données

Remarquez en particulier que l'attribut **architecture** de l'élément 'dbConnection' est positionné à '**clientDatabase**'. Ceci signifie que l'application sera générée en un bloc. Vous pouvez aussi choisir '**clientServerDatabase**' et une servlet sera générée, en plus de votre application.

Deuxième partie : Interface Utilisateur.

Ici est décrit l'apparence du logiciel que l'on souhaite générer. C'est la partie la plus complexe à écrire, mais aussi la plus intéressante.

Les composants disponibles :

Une interface graphique est un ensemble d'écrans (**screen** en anglais). Dans chaque écran, on trouvera des formulaires (**form**). De plus, certains formulaires contiennent des champs (**field**). Ce qui nous fait trois niveaux de composants.

- **Screen** : il n'en existe qu'une seule sorte.
- **Forms** : on en utilisera quatre.
 - **menuForm** : Des boutons pour ouvrir d'autres screens.
 - **listForm** : Pour afficher et imprimer le résultat d'une requête.
 - **cardForm** : N'affiche qu'un enregistrement à la fois. C'est une fiche qui contient un ensemble de champs.
 - **researchForm** : Pour faire des recherches.
- **Fields** : on en utilisera deux.
 - **textField** : Affiche une valeur, sous forme d'une ligne de texte.
 - **comboBoxFkField** : Affiche les colonnes de l'identifiant utilisateur (**userKey**) de la table référencée. Ce champ s'applique à une clef étrangère.

Composant de GUI : le menuForm

Le premier écran qui s'affichera à l'utilisateur sera un menu qui permettra d'accéder aux autres écrans. On a donc un **screen** contenant un **menuForm**.

Le menuForm est une simple liste de boutons qui ouvrent sur des screens.

Je vous donne ici le code du menu principal.

On remarque que le formulaire référence quatre screens : SLFle, SCFle, SRCmd, SCCmd.

Il est donc nécessaire de créer ces quatre autres screens. Je vous propose de commencer par SLFle.

```
<screen name='SMFleCmd'>
  <title></title>
  <formSet>
    <menuForm>
      <menuTitle height='70'>Base des Fleurs
      </menuTitle>
      <firstButtonLocation x='100' y='150' />
      <buttonSize w='400' h='36'
        verticalGap='18' />
      <menuButtonList>
        <menuButton screen='SLFle'>
          Liste des Fleurs</menuButton>
        <menuButton screen='SCFle'>
          Ajout de Fleurs</menuButton>
        <menuButton screen='SRCmd'>
          Recherche de commandes</menuButton>
        <menuButton screen='SCCmd'>
          Ajout de Commandes</menuButton>
      </menuButtonList>
    </menuForm>
  </formSet>
</screen>
```

GUI : Le menu principal

Base des Fleurs

Liste des Fleurs

Ajout de Fleurs

Recherche de commandes

Ajout de Commandes

Le screen SMFleCmd

Composant de GUI : le listForm

L'écran 'SLFle' affiche la liste des fleurs, et pour chaque fleur, le nombre de commandes. On a donc un **screen** contenant un **listForm**.

Le listForm se base sur une table définie dans l'environnement. Ici la table 'tblfle', qui correspond à la table réelle FLEUR.

Après la table principale, on décrit les colonnes à afficher. Nous en voyons deux sortes : une **instanceColumn** correspond à une colonne réelle de la table.

Une **agregatInstanceColumn** correspond à une fonction d'agregat dans la clause SELECT, pour une table dépendante. La nôtre travaille sur la clef étrangère 'fk_cmd_fle' définie dans l'environnement.

```
<screen name='SLFle'>
  <title>Liste des Fleurs</title>
  <formSet>
    <listForm db='dbFleur'
      doubleClicScreen='SCFle'
      insertScreen='SCFle'
      delete='yes'>
    <bounds x='10' y='10' w='500' h='300' />
    <schemaTableRef schemaTable='tblfle' />
    <instanceColumnList>
      <instanceColumn
        schemaColumn='tblfle_Libelle'
        letterSpacing='36' sort='asc' />
      <agregatInstanceColumn
        schemaFk='fk_cmd_fle' type='count'
        letterSpacing='5' sort='desc'
        <title>Nb Cmd</title>
      </agregatInstanceColumn>
    </instanceColumnList>
  </listForm>
</formSet>
</screen>
```

GUI : La liste de fleurs

On remarque que le formulaire référence le screen SCFle par deux fois. Ce dernier était de plus référencé dans le menuForm de la section précédente. Voyons comment créer ce screen.

Liste des Fleur(s)

Type de fleur	Nb Cmd
Anémone	12
Crocus	2
Fuchsia	0
Iris	22
Jonquille	12
Lys	14
Muguet	2
Oeillet	20
Orchidée	14
Rose blanche	9
Rose orange	25
Rose rouge	11

Nouveau 12

Le screen SLFle

Instances de colonnes

Une **'instanceColumn'** représente la colonne d'une instance de table (une table surnommée) dans la clause FROM de la requête SQL sous-jacente.

Lorsqu'on ne précise pas l'instance de table, l'application utilise celle par défaut de la table physique définie dans l'environnement.

Composant de GUI : le cardForm

L'écran 'SCFle' affiche la fiche d'une seule fleur à la fois, c'est-à-dire toutes les informations la concernant. De plus, à chaque Fleur est associé un ensemble de Commandes. On a donc un **screen** contenant un **cardForm** et un **listForm**.

Le cardForm travaille sur une table définie dans l'environnement. Ici la table 'tblfle', qui correspond à la table réelle FLEUR. Ensuite vient la description des champs (fields).

Le field **textField** affiche une valeur sous forme de texte. Ici, c'est la valeur correspondant à la colonne tblfle_Libelle (la colonne réelle Fleur.Libelle).

Je vous passe le contenu du listForm des commandes, reportez-vous au fichier fleur.xml.

```
<screen name='SCFle'>
  <title>Fiche d'une fleur</title>
  <formSet>
    <cardForm db='dbFleur'>
      <title>Fiche d'une Fleur</title>
      <location x='10' y='10' />
      <cardSchemaTableRef
        schemaTable='tblfle'
        updateAfterInsert='yes'
        multipleInsert='no' queries='sudi' />
      <fieldList>
        <textField>
          <schemaColumnRef schemaColumn=
            'tblfle_Libelle' />
        </textField>
      </fieldList>
    </cardForm>
    <listForm db='dbFleur'>
      <!-- ... liste des commandes ... -->
    </listForm>
  </formSet>
</screen>
```

GUI : La Fiche d'une Fleur

Fiche d'une Fleur

Type de fleur: Rose blanche

Supprimer Enregistrer

Liste des Commande(s)

Livraison	Type de fleur
12-11-2002	Rose blanche
03-11-2002	Rose blanche
02-11-2002	Rose blanche
28-10-2002	Rose blanche
17-10-2002	Rose blanche
10-10-2002	Rose blanche
08-10-2002	Rose blanche
05-10-2002	Rose blanche
03-10-2002	Rose blanche

Nouveau 9

Le screen SCFle : un cardForm et un listForm

Composant de GUI : le researchForm

Le menu permettait aussi d'ouvrir un écran de recherche de commandes. Voyons comment le construire. On aura un **screen** contenant un **researchForm**. Ce dernier est composé d'une liste de champs (fields) ainsi que d'un listForm pour afficher le résultat. Que du déjà vu !

On remarque dans le code ci-contre le champ **'comboBoxFkField'** qui affiche une valeur de clef étrangère. On retrouvera le même dans la fiche (cardForm) de la commande (écran SCCmd, à voir dans fleur.xml).

```
<screen name='SRCmd'>
  <title>Recherche de commandes</title>
  <formSet>
    <researchForm>
      <title>Recherche de commandes</title>

      <fieldList>
        <comboBoxFkField>
          <schemaFkRef schemaFk='fk_cmd_fle' />
        </comboBoxFkField>
        <textField>
          <schemaColumnRef schemaColumn=
            'tblcmd_DateLivraison' />
        </textField>
      </fieldList>

      <listForm db='dbFleur'>
        <!-- ... liste des commandes ... -->
      </listForm>

      <researchFormCriteriaBuilder mode='and'
        defaultSubEmptyAction='ignore'
        emptyAction='stuck' />
    </researchForm>
  </formSet>
</screen>
```

GUI : Recherche de Commandes

Livraison	Type de fleur
03-11-2002	Anémone
03-11-2002	Iris
03-11-2002	Jonquille
03-11-2002	Lys
03-11-2002	Orchidée
03-11-2002	Rose blanche
03-11-2002	Rose orange
03-11-2002	Rose rouge

Le screen SRCmd

SURPRISE ! Avez-vous remarqué ? Il manque le lieu de la livraison ! Exercice : rajouter le lieu de la livraison...

Comment générer l'application

Il vous faut avant tout installer le **melbalab** (à télécharger sur www.sashipamelba.com). Voici les étapes d'installation :

- Installez le **J2SDK 1.4** (i.e. JDK) de Sun Microsystems, si ce n'est déjà fait.
- Dézippez le melbalab dans un répertoire pas trop difficile à accéder depuis le shell. (Sous Windows, mettez-le proche de la racine).
- Editez le fichier `<melbalab-home>\melbabuild.bat` (Windows) ou `<melbalab-home>/melbabuild` (Linux), puis définissez la variable `JAVA_HOME_BIN`. Si vous souhaitez générer des servlets, définissez aussi `JAVAX_JAR`.
- SGBD : vérifiez que votre SGBD est prêt, et que les pilotes ODBC ou JDBC sont installés.

Maintenant, vous êtes prêt à générer votre application.

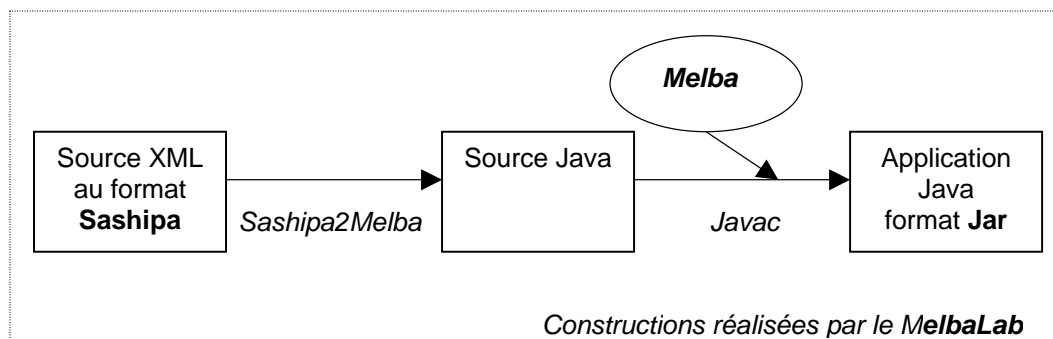
- Copiez le fichier « fleur.xml » vers le répertoire `<melbalab-home>/src_xml` s'il n'y est pas déjà.
- Editez-le et modifiez l'élément **dbConnection** (en fin de fichier) pour pointer vers votre SGBD.
- Ouvrez un shell (une boîte DOS)
- Allez dans le répertoire `<melbalab-home>/` (commande `cd <nom-repertoire> ...`)
- Saisissez **melbabuild fleur** .

Votre application générée est le fichier **fleur.jar**, qui est déposé dans `<melbalab-home>/result` . Exécutez-la en double-cliquant dessus ou avec la commande : **java -jar result/fleur.jar** .

Les coulisses...

Heu... je plaisantais pour le lieu de livraison.

Un petit schéma pour résumer les étapes de la construction avec Sashipa-Melba :



Votre fichier XML est traduit en source Java (que vous pouvez visualiser dans le répertoire `<melbalab-home>/work/src_java/` , après construction). Le source Java ainsi généré ne contient que des appels de la bibliothèque de programmation Melba. Melba est le gros moteur Java qui contient toute la logique de votre application.

Javac est le compilateur Java du J2SDK (Sun Microsystems).

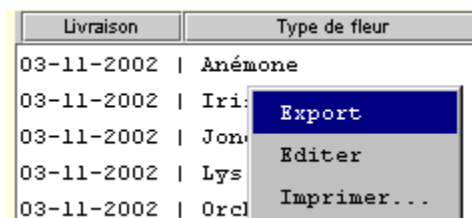
Le fichier que génère en fin de compte le MelbaLab est un fichier jar (Java Archive). Sachez qu'un fichier 'jar' est en fait un fichier Zip. Il vous suffit de renommer l'extension en '.zip' pour pouvoir le dézipper, s'il vous prend l'envie de jeter un coup d'œil dedans.

Votre application ainsi générée est une application Java. Ceci implique qu'il faut une **machine virtuelle Java** pour pouvoir l'exécuter. Vous trouverez celle de Sun Microsystem dans le JRE, téléchargeable gratuitement. Le JRE est inclus dans le J2SDK que vous avez forcément déjà installé sur votre machine, mais pensez à l'installer sur les machines de vos utilisateurs.

Petit tour d'horizon

Etats et exports

Depuis le menu contextuel de chaque listForm, vous avez accès à ses états disponibles. Chaque listForm dispose d'un état par défaut sans que vous l'ayez défini. Il est bien sûr recommandé d'en créer d'autres comme le fait l'application DemoContact du MelbaLab. Depuis ce même menu contextuel, vous pouvez exporter vos données vers un fichier texte ou par copier/coller (dans Excel par exemple).



Menu contextuel des listForm

Licences

Les éléments de la technologie Sashipa-Melba sont tous sous licences publiques. Les applications générées avec sont bien sûr votre propriété, pour des buts commerciaux ou non. Attention en revanche à la licence du J2SDK de Sun.

Architectures générées

Un apport de Sashipa-Melba qui vous changera la vie, c'est la facilité avec laquelle vous pouvez changer et adapter l'architecture de votre application.

Voici les architectures disponibles :

- La partie GUI (cliente) peut être une **application** ou une **applet**.
- La partie serveur peut être une **servlet**, ou peut être **incorporée dans la GUI**. Dans le cas d'une servlet, les fonctionnalités **multi-utilisateurs** sont activées avec les verrous sur enregistrements et le rafraîchissement des caches sur tous les clients.
- L'accès aux bases de données se fait via **ODBC**, **JDBC**, ou par une **page Web dynamique** telle qu'une PHP (pratique lorsque votre hébergeur ne propose pas les servlets !).
- Les logiciels générés sont en Java. Donc **portables** sur la plupart des OS.
- **Un nombre croissant de SGBD** sont supportés.

Vous trouverez la combinaison la plus adaptée à vos besoins. Tous ces choix structurels sont de toutes façons faciles à changer en fin de développement.

Sashipa-Melba vous changera la vie !

Sashipa-Melba vous changera la vie, à vous et vos utilisateurs. Vous pourrez enfin leur proposer ce qu'ils vous ont toujours réclamé :

- Comparé à une solution programmée, vous divisez par quelques dizaines le temps entre la détermination du besoin et la livraison.
- Les applications obéissent toutes à la même logique de présentation. Le temps d'apprentissage est réduit dès la deuxième application livrée.
- Vos applications sont sans bugs du premier coup. (Il en reste extrêmement peu du fait que plusieurs applications utilisent le même moteur)
- Pas d'effets de bords quand vous rajoutez une fonctionnalité. De plus, le source n'en devient pas plus complexe, il grossit proprement.
- Vous pouvez consacrer l'essentiel du temps alloué au projet à l'analyse, et non au développement.

Ne développez plus !

Depuis quelques jours, un programme nommé Database2Sashipa génère pour vous un fichier source en Sashipa, à partir d'une connexion à votre base de données et d'hypothèses par défaut. Désormais, il est donc possible de créer une application en Sashipa en quelques secondes ! De plus, le fichier généré est très propre et vous servira de base pour créer l'application de vos rêves... Testez sur votre base !

Une technologie qui évolue...

Je travaille actuellement sur l'internationalisation, je pense finir en novembre prochain. En novembre Sashipa-Melba sort de France ! Une communauté de développeurs se constitue en ce moment, les évolutions s'accélèrent. Mettez-vous à jour régulièrement sur www.sashipamelba.com ! .

Thomas (thomas.mur@sashipamelba.com)

Sommaire

SGBD : DEVELOPPEZ D'ABORD, CHOISISSEZ APRES !.....	1
PRESENTATION	1
PRELIMINAIRE : CREATION DE LA BASE DE DONNEES	1
APPRENTISSAGE DE SASHIPA	2
<i>Première partie : Environnement.....</i>	<i>2</i>
<i>Dernière partie : Architecture.</i>	<i>3</i>
<i>Deuxième partie : Interface Utilisateur.</i>	<i>3</i>
COMMENT GENERER L'APPLICATION	8
LES COULISSES... ..	8
PETIT TOUR D'HORIZON	9
<i>Etats et exports</i>	<i>9</i>
<i>Licences</i>	<i>9</i>
<i>Architectures générées.....</i>	<i>9</i>
<i>Sashipa-Melba vous changera la vie !</i>	<i>9</i>
<i>Ne développez plus !</i>	<i>10</i>
<i>Une technologie qui évolue... ..</i>	<i>10</i>
SOMMAIRE.....	10