

DATAOBJECT ET ACCÈS AUX BASES DE DONNÉES : UNE INNOVATION NAT JET

Le principal travail d'une application «métier» est la transmission d'informations entre une base et un écran.

Pour cela, il faut écrire des ordres SQL pour alimenter un bean Java puis écrire du code pour copier les informations du bean dans des contrôles.

Une fois la saisie réalisée, on repart dans le sens inverse : de l'écran vers la base de données.

Entre temps, on a peut être écrit quelques règles et contrôles, mais une partie importante du travail est un travail fastidieux de copie entre différents éléments ou structure : contrôle graphique, structure mémoire ou colonne d'un enregistrement en base.

NatJet dispose de trois éléments qui prennent en charge cette problématique :

- NatOrb : il s'agit d'une solution de mapping relationnel/objet qui s'appuie sur Hibernate. Le développeur ne connaît plus la structure base, il ne manipule que sa vision logique (les beans). Le développeur n'écrit plus les ordres SQL, ceux-ci sont générés par le couple NatOrb/Hibernate. Le passage de la structure mémoire (le bean Java) vers l'enregistrement base est pris en charge par NatOrb,
- DataObject : cet objet fait le pont entre la structure mémoire (le bean Java) et les contrôles graphiques de la page,
- ListView : cet objet fait le pont entre une collection d'objets et un contrôle de type Liste.

NatOrb : la solution de mapping relationnel/objet

NatOrb est une surcouche d'Hibernate dont la grande force est son intégration avec NatJet : NatJet sait quand une requête http arrive et repart. Il prévient NatOrb, ce qui permet à ce dernier de prendre en charge les aspects les plus techniques de l'utilisation d'une solution de mapping relationnel/objet comme la libération des connexions utilisées par le développeur et le détachement des objets conservés en mémoire.

Grâce à ce mécanisme, NatOrb garantit qu'un développeur n'utilisera pas plus d'une connexion base de données par requête http et que celle-ci sera automatiquement libérée à la fin de la requête http. Vous êtes ainsi protégé

d'une éventuelle consommation excessive de ressources base de données par vos développeurs et donc de l'effondrement des performances de vos applications lors de la mise en production.

On retrouve également tous les avantages de l'utilisation d'une solution de mapping relationnel/objet comme Hibernate :

- Le développeur n'a plus besoin d'écrire des ordres SQL,
- Utilisation d'un pool de connexion base permettant une optimisation des ressources base de données,
- Possibilité d'utiliser simplement des requêtes de types agrégation comme la somme, le minimum...
- Possibilité de forcer des jointures afin d'optimiser les requêtes SQL et de remonter en une seule requête SQL plusieurs objets,
- Possibilité d'utiliser des procédures stockées...

DataObject et son Editeur

Le DataObject est un composant invisible qui s'ajoute dans le Graphical Designer à un panneau/écran. Il possède deux niveaux de fonctionnalités :

- Faire le lien entre un bean et des contrôles graphiques de l'écran,
- Faciliter l'accès à la base de données.

La fonctionnalité première du DataObject est, grâce à son éditeur, de faire le lien entre les contrôles de l'écran et les propriétés d'un bean Java. Ce bean Java est un bean quelconque : il peut provenir d'un EJB Session, d'un appel à un ServiceWeb ou de toute solution de mapping/relationnel objet... Cette liste n'est pas exhaustive.

Rappel sur les beans Java

Pour les lecteurs peu familiers avec les concepts Java : le bean représente souvent un objet métier Java. Il ne doit respecter qu'une contrainte :

- Il existe des méthodes appelées getter/setter qui permettent d'accéder à ses propriétés. Ces méthodes respectent une règle de nomenclature : nom de la propriété en majuscule préfixé de get ou set. Ces méthodes peuvent être générées automatiquement depuis un IDE comme Eclipse et donc depuis NatJet. En règle générale, le bean a besoin d'être «serialisable» (c'est particulièrement

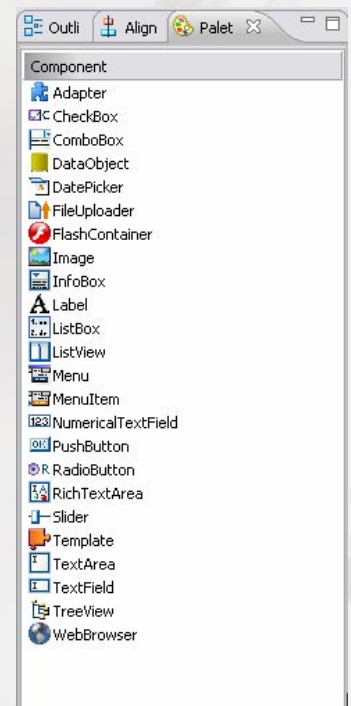
important pour les applications Web), il doit donc également :

- Posséder un constructeur vide afin de permettre la sérialisation de l'objet,
- Implémenter l'interface «serialisable» qui ne demande rien d'autre qu'un constructeur vide.

Le bean est la structure Java de base pour diverses solutions : EJB, Service Web, Hibernate...

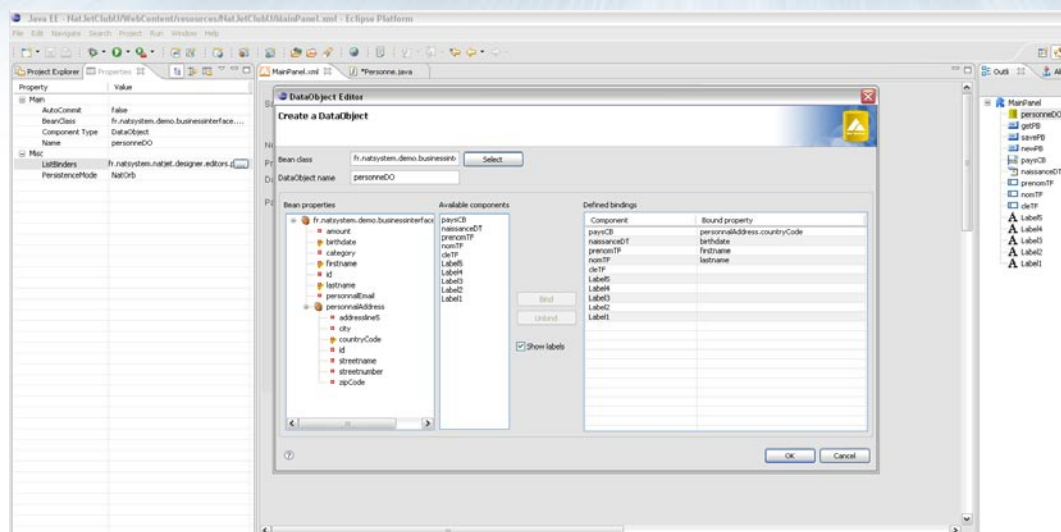
Le DataObject Editor

Comme nous l'avons déjà dit, l'éditeur permet de choisir un bean ainsi que les beans liés afin de les associer aux contrôles d'un panneau. Pour cela, on édite son panneau (l'équivalent d'une fenêtre NS-DK) avec le Graphical Designer de NatJet. Depuis la palette, on ajoute le contrôle DataObject Editor en le glissant/déposant sur le panneau.



Notez, le clin d'œil fait aux utilisateurs de NatStar : le DataObject reprend la même signalétique que les Data Templates : les fameux «bidons jaunes».

Le DataObject étant un contrôle non graphique, il n'apparaît pas dans le panneau afin de ne pas obstruer la vision du design, il faut passer par la vue Outline afin de le sélectionner.



propriété PersistenceMode à NatOrb dans le volet Properties.

Ce mode rend accessibles les méthodes loadFromDatabase et saveToDatabase du DataObject.

Voir les exemples de Get et Save en Mode NatOrb.

Le DataObject utilise NatOrb pour accéder à la base de données mais ce n'est pas le seul service qu'il rend : tout objet provenant d'une solution comme Hibernate ne peut pas être stocké simplement dans une session J2ee. Il est impératif de «Détacher» l'objet quand on quitte le serveur et de le «ReAttacher» avant toute tentative de mise à jour.

Ces opérations alourdissent le code et le rendent très technique. Le couple DataObject/NatOrb prend en charge ces opérations techniques simplifiant encore plus le code.

Conclusion

Au cours de cette fiche technique nous avons pu voir la puissance et la facilité de développement apportée par le couple DataObject/NatOrb pour les accès à une base de données :

- La quantité de code à écrire est très faible,
- Le code est extrêmement simple,
- NatOrb s'assure que le développeur ne consomme pas plus de ressource base de données (notamment les connexions) que nécessaire et s'assure que l'application puisse monter en charge.

Grace à l'intégration forte entre Hibernate et NatJet assurée par le couple NatOrb/DataObject la construction d'écran alimenté par des données d'une base devient extrêmement simple.

Quand sa propriété ListBinders est sélectionnée, il est possible d'appuyer sur un bouton permettant d'ouvrir son éditeur.

Dans l'éditeur, on choisit la classe Java correspondant au Bean, il est alors possible de faire correspondre les propriétés du bean au contrôle du panneau.

Quand une propriété a été «bindée» (reliée à un contrôle graphique), son icône est modifiée afin d'assurer un meilleur suivi.

Les utilisateurs de NatStar noteront que contrairement à NatStar, le DataObject permet d'utiliser des contrôles existant d'une maquette : il est ainsi possible de faire, dans un premier temps, une maquette des écrans de son application. Une fois que cette maquette a été validée, il est possible de construire son modèle de données et de faire le lien entre les contrôles et ce modèle de données sans devoir redessiner son écran. La maquette devient l'application finale.

La programmation du DataObject

Une fois le DataObject créé et défini, il est possible de lui affecter un Bean (l'instance de l'objet) par programmation, afin d'alimenter le contenu des différents contrôles associés au DataObject.

Il est également possible de remplir un Bean avec les valeurs des contrôles du panneau en utilisant la méthode saveToBean(monBean).

Ces deux méthodes seront utilisées si on travaille avec des EJB ou des Services Web.

Dans l'exemple suivant, on recherche une instance en utilisant un Service Web et on l'affiche grâce au DataObject.

```
@Override
public void getPB_ExecutedEvent(
    NsExecutedEvent arg0) {
    // Initialisation JAX-WS du client de
    // serviceWeb
    WebServicePersonne monSw =
        new WebServicePersonne();
    ServicePersonne servicePersonne =
        monSw.getServicePersonneImplPort();

    // Appel de le méthode du service web
    Personne personneTrouvee =
        servicePersonne.getPerson(
            cleTF.getText());

    // Alimenter les contrôles avec les
    // valeurs des propriétés du bean
    // Personne récupéré précédemment.
    personneDO.loadFromBean(
        personneTrouvee);
}
```

DataObject et Accès base de données

Pour accéder à la base de données, vous pouvez utiliser les APIs de NatOrb et les deux API précédentes du DataObject. Néanmoins, vous vous rendez vite compte que ce code est générique, c'est pourquoi nous

avons décidé, afin de simplifier encore plus le développement, de proposer des APIs spécifiques à ce mode de fonctionnement directement dans le DataObject.

Pour pouvoir utiliser ce mode de fonctionnement, il faut au préalable informer le DataObject qu'il devra fonctionner dans ce mode en positionnant simplement sa

```
@Override
public void getPB_ExecutedEvent(NsExecutedEvent arg0)
{
    try {
        personneDO.getFromDatabase(cleTF.getText());
    } catch (DataObjectIdNotFoundException e) {
        msgBox("La clé ne correspond à aucune personne",
            "Avertissement");
    }
}
```

Exemple de Get en Mode NatOrb

```
@Override
public void savePB_ExecutedEvent(NsExecutedEvent arg0)
{
    try {
        personneDO.saveToDatabase();
    } catch (DataObjectException e) {
        msgBox("La mise à jour n'est pas possible.",
            "Erreur");
    }
}
```

Exemple de Save en mode NatOrb