Microsoft planche sur un OQL pour sa plateforme .Net 3.0

par Silver Nakache 03/09/2006

J'ai toujours été étonné du retard de Microsoft sur ce point : un OQL (Object Query Language) est un élément déterminant des architectures objets dès que l'on est confronté à un problème de persistance. Il s'agit d'un langage, au même titre que SQL par exemple, spécifié par l'ODMG (Object Data Management Group) destiné à opérer toutes les opérations de sélection, de requêtage et de mise à jour en manipulant des objets natifs.

Sans polémiquer, un tel langage manquait cruellement aux plateformes de développement Microsoft. Alors que des produits comme Versant pour plateforme Java existent depuis bien longtemps déjà. Il est vrai que leur performance laissait à désirer mais cela se travaille.

Pour ma part, je guettais désespérément la moindre couche objet-db sur SQL Server, qui transformerait ce DBMS en ODBMS (Object Data Base Management System), mais rien.

Heureusement, dès que Microsoft tarde à combler ses manques, nous avons une panoplie de technologie qui se développe. Je pense notamment à des technologies comme NHibernate entre autre.

Pourquoi un OQL est-il si important ?

Lors de la conception d'une application basée sur les objets, je vois trop souvent les développeurs commencer par une modélisation de la base de données. Or, dans une organisation à base d'objets, il est primordial de ne pas être dicté par un moyen de persistance, car dans ce sens, c'est la base de donnée qui dicte la formation des objets à l'architecture. Pour garder toute l'agilité nécessaire, les objets (classes) doivent pouvoir coller au plus prêt au métier, doivent pouvoir évoluer, être 'refactorer' au fur à mesure des évolutions du projet. Un autre aspect important est la capacité d'un 'Persistance Provider', c'est à dire en deux mots, pouvoir changer de base de données. Pourquoi mettre à la Corbeille son architecture objet lorsque l'on souhaite changer de persistance ?

Outre les considérations d'indépendance vis-à-vis d'un fournisseur, je vois dans ces méthodes de requêtage objet une étape supplémentaire dans la simplification de la programmation pour coller au plus prêt de la demande business.

A quoi ressemble un requêtage d'objet natif en LINQ :

Etape préliminaire, créer l'objet :

```
public class Person {
  string name;
  int age;
  bool canCode;

  public string Name {
    get { return name; } set { name = value; }
  }

  public int Age {
    get { return age; } set { age = value; }
  }

  public bool CanCode {
    get { return canCode; } set { canCode = value; }
  }
}
```

Instancier une liste d'objet :

```
static Person[] people = {
  new Person { Name="Allen Frances", Age=11, CanCode=false },
  new Person { Name="Burke Madison", Age=50, CanCode=true },
  new Person { Name="Connor Morgan", Age=59, CanCode=false },
  new Person { Name="David Charles", Age=33, CanCode=true },
  new Person { Name="Everett Frank", Age=16, CanCode=true },
};
```

Créer une requête Select les objets :

```
var expr = people.Select(p => new {
               p.Name, BadCoder = p.Age == 11
           });

foreach (var item in expr)
  Console.WriteLine("{0} is a {1} coder",
                    item.Name,
                    item.BadCoder ? "bad" : "good");
```

Dans cet exemple, notre « Persistance Provider » est en fait la mémoire de notre PC.

Regardons maintenant un exemple avec « Persistance Provider » orienté Database (DLINQ) :

Définition des objets :

```
[Table(Name="People")]
public class Person {
  [Column(DbType="nvarchar(32) not null", Id=true)]
  public string Name;

  [Column]
  public int Age;

  [Column]
  public bool CanCode;
}

[Table(Name="Orders")]
public class Order {
  [Column(DbType="nvarchar(32) not null", Id=true)]
  public string OrderID;

  [Column(DbType="nvarchar(32) not null")]
  public string Customer;

  [Column]
  public int? Amount;
}
```

A noter les attributs [Table] et [Column] qui permettront la génération de code intermédiaire pour la base de données configurée (SQL Server par exemple).

Requêtage des objets :

```
// establish a query context over ADO.NET sql connection
DataContext context = new DataContext(
     "Initial Catalog=petdb;Integrated Security=sspi");

// grab variables that represent the remote tables that
// correspond to the Person and Order CLR types
Table<Person> custs = context.GetTable<Person>();
Table<Order> orders   = context.GetTable<Order>();

// build the query
var query =
    from c in custs
    from o in orders
    where o.Customer == c.Name
    select new {
        c.Name, o.OrderID, o.Amount, c.Age
    };

// execute the query
foreach (var item in query)
    Console.WriteLine("{0} {1} {2} {3}",
        item.Name, item.OrderID, item.Amount, item.Age);
```

Et cela fonctionne virtuellement avec n'importe quel « Persistance Provider ». Avec XML par exemple, Microsoft le baptise WLINQ.

Ce que le projet LINQ n'adresse pas pour l'instant c'est:

1- Le difficile problème du versioning des objets : que se passe-t-il quand on ajoute ou retire une donnée membre d'un objet avec sa persistance ?
2- Comment faire suivre la persistance lorsque le modèle change ?

A suivre attentivement …

Liens externes :
Le projet LINQ de Microsoft : http://msdn.microsoft.com/data/ref/linq/default.aspx
La spécification OQL du DOMG : http://www.odmg.org/
Versant ODBMS : http://www.versant.com/en_US/products/objectdatabase
NHibernate for .NET: http://www.hibernate.org/