BE3 - Architecture logicielle

Wael Ben Soltana

2012 - 2013

1 Description du BE

Le principe de ce BE est de continuer la réalisation du projet en s'occupant de l'architecture logicielle : un système complet avec spécifications des fonctionnalités selon le sujet choisi. **La manipulation avec la base de données est par conséquent invisible aux utilisateurs**.

2 Travail attendu

Après la conception et la réalisation de la base de données (tables, relations, procédures stockées, déclencheurs), il est temps de construire un programme permettant l'interaction entre la base et les utilisateurs. On attend pour cela que vous remplissiez des tâches suivantes :

- **Choix du langage de programmation** (PHP, Java, C++, Visual C++ .Net...)
- **Définition des interfaces graphiques selon les fonctionnalités** (la seule partie visible aux utilisateurs)
- **Manipulation d'une base de données depuis du code** (en fonction de SGBD et logiciel de programmation choisie)

3 Manipulation d'une base de données depuis du code

Le principe de base est le suivant :

- On établit une connexion
- On effectue des requêtes retournant (ou non) des ensembles résultats (qui sont des collections parcourables, selon le langage, directement ou le plus souvent via un itérateur)
- Puis on ferme la connexion

Ce modèle d'accès est globalement indépendant du SGBD du mode d'accès choisi ou du langage de programmation.

Nous allons détailler un peu le problème de la connexion qui est le plus susceptible de poser problème

- La connexion peut s'effectuer soit via le code, soit via un objet externe (fichier de connexion, lien ODBC). D'une manière ou d'une autre, vous devez disposer de connecteurs (drivers) adaptés à votre système de gestion de bases de données. Ceux-ci peuvent être intégrés dans votre environnement de programmation voire au système d'exploitation (sans surprise vous n'avez rien à installer pour accéder à Access ou SQL Server depuis Windows). Les drivers ODBC sont toutefois toujours gratuits et en libre téléchargement.

– **Une connexion via le code** va se faire via des connecteurs internes (ADO avec .Net, JDBC avec Java), ici il n'est pas possible de donner des explications non spécifiques, chaque langage ayant sa propre syntaxe. Toutefois il s'agit a priori des connecteurs les plus performants.

– En ce qui concerne les connexions externes

- **ODBC** (Open Database Connectivity) est un standard développé et imposé par Microsoft pour permettre **n'importe quel programme de communiquer avec n'importe quelle base**.

- **JDBC** (Java Database Connectivity) est une interface de programmation (API) fournie avec Java (depuis sa version 1.1) permettant de se connecter à des bases de données, c'est-à-dire que JDBC constitue un ensemble de classes permettant de développer des applications capables de se connecter à des serveurs de bases de données (SGBD).

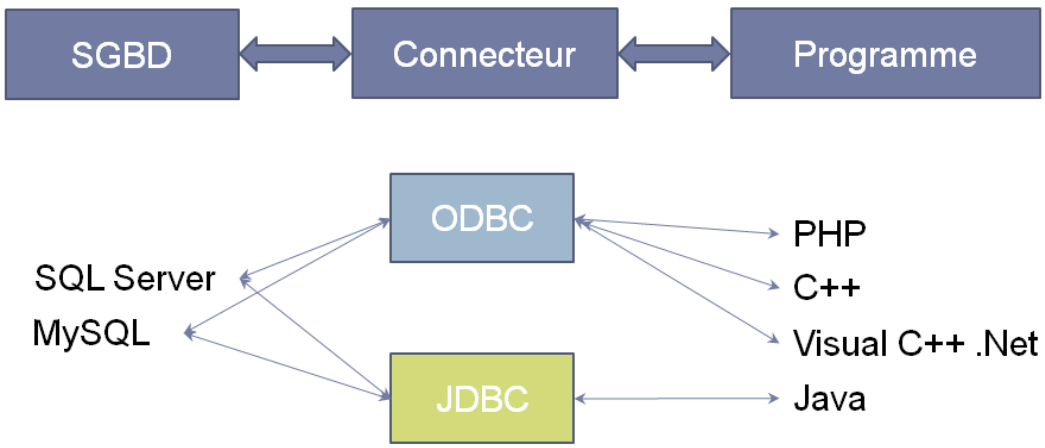

FIGURE 1 – Schéma de connexion d'un programme à une base de données.

En pratique, il y a de différents connecteurs et des nuances dans le code en fonction du SGBD et du langage de programmation. Vous trouvez ci-dessous des liens utiles favorisant la réalisation de votre programme. **Ces liens peuvent être accédés directement depuis la version .pdf de ce sujet qui se trouve à http ://liris.cnrs.fr/∼cdnguyen/BEBD3A/BE3 - sujet.pdf**

- **PHP / ODBC / (MySQL, SQLServer) MySQL Connector/ODBC 5.1**
  http ://dev.mysql.com/downloads/connector/odbc/5.1.html

  Accès à la base de données
  http ://php.net/manual/fr/book.uodbc.php
  http ://www.w3schools.com/PHP/php_db_odbc.asp

  Appel d'une procédure stockée
  http ://dev.mysql.com/doc/refman/5.0/en/call.html
  http ://www.php.net/manual/fr/function.mssql-execute.php
  http ://it.toolbox.com/wiki/index.php/Running_a_Stored_Procedure_using_PHP_and_ODBC

  Création d'un déclencheur depuis PHP
  http ://www.lephpfacile.com/manuel-mysql/triggers.php

- **C++ / ODBC / (MySQL, SQLServer)**
  http ://simpledb.sourceforge.net/
  http ://www.sqlapi.com/
  http ://devmentor-unittest.googlecode.com/files/ODBC%20Programming%20in%20C%2B%2B.pdf

- **Java / JDBC / MySQL**
  JDBC Connector/J 5.1
  http ://dev.mysql.com/downloads/connector/j/5.1.html

  Accès à la base de données
  http ://tecfa.unige.ch/guides/tie/pdf/files/java-mysql.pdf
  http ://www.developer.com/java/data/article.php/3417381/Using-JDBC-with-MySQL-Getting-Started.htm
  http ://www.stardeveloper.com/articles/display.html ?article=2003090401&page=1

  Appel d'une procédure stockée
  http ://www.java2s.com/Tutorial/MySQL/0201__Procedure-Function/CallingastoredprocedureinJava.htm

- **Java / JDBC / SQL Server**
  Connecteur JDBC 1.2 SQL Server
  http ://www.microsoft.com/downloads/details.aspx ?FamilyId=C47053EB-3B64-4794-950D-81E1EC91C1BA

  Manipulation de la base de données
  http ://msdn.microsoft.com/fr-fr/architecture/cc440514.aspx
  http ://download.microsoft.com/download/0/d/b/0db5b099-c24f-4b0f-a132-fb287b326ee2

- **C++ .net / ODBC / (MySQL, SQLServer)**
  Manipulation de la base de données : voir la section 5

  Procédure stockée
  http ://support.microsoft.com/kb/310142/fr

4 Règles générales de programmation

1. Toute personne nommant avec a,b,c... data1, data2 ou tout autre nom générique flou des variables qui ne sont pas des compteurs et dont la portée dépasse un bloc de 3 lignes sera froidement assassinée.
2. On ne crée pas une fichier MightyThingThatDoesAll dans laquelle on gère l'interface graphique, les accès Base de Données et qui calcule des décimales de pi quand le pc n'a rien à faire (principe d'encapsulation, réutilisation... bref des avantages de la programmation objet - pour la liste complète cf examen de POO 2008-2009). En pratique isoler les composants les rend plus faciles à débugger, c'est une bonne habitude à prendre, donc par exemple on fait une classe pour chaque fenêtre (et plus généralement pour chaque fonctionnalité) qui ne servira rien qu'à ça, merci.
3. En théorie un programme ne doit planter sous aucun prétexte (la "license to crash" est une exclusivité réservée aux professionnels) les erreurs de l'utilisateur devraient être traitées et générer un message d'erreur. Bien sûr étant donné le temps disponible vous pouvez vous contenter de mentionner les cas d'erreur connus dans votre compte rendu.

5 Une application utilisant C++ .net

Si vous n'avez aucune expérience avec un autre langage de programmation nous vous proposons une solution via ODBC/Visual C++ .net

Ce document (comme l'interface logicielle mise à votre disposition) a été conçu pour vous permettre de coder une application en utilisant vos connaissances de C++ des 2 premières années via un minimum d'adaptation.

5.1 Introduction, un peu de culture générale .net

C++ étant antérieur aux interfaces graphiques, la réalisation d'une interface graphique sous C++ n'est pas triviale et nécessite l'utilisation d'une série de bibliothèques. Plus particulièrement sous l'environnement Visual Studio ceci se faisait via l'utilisation de la tristement célèbre MFC (pour Microsoft Foundation Class) renommée pour sa complexité. Le passage à .net a en revanche introduit un nouveau modèle (Windows Forms) qui permet de **réaliser très simplement des interfaces graphiques complètes**. La mauvaise nouvelle (car il y en a forcément une) est du côté du code puisque windows forms est écrit en C++ CLR, une version du langage C++ spécifique à l'environnement .net et qui diffère pas mal de ce que vous avez appris ces dernières années.

Le CLR (Common Language Runtime) est un environnement d'exécution qui est commun aux langages .net : C#, C++ CLR, VB .net et J#, le code ainsi écrit n'est plus directement traduit en langage machine mais en **un langage intermédiaire commun** (IL, Intermediate Language) ce qui facilite les choses pour les bibliothèques puisque du coup si on les écrit en IL, ce sera les mêmes pour tout les langages. Problème : le IL travaille avec des outils absents du C++ standard : garbage collector, contrôle des collections, etc. D'où les nouvelles notations et instructions que vous allez trouver dans le code généré dans votre application : tout ceci vise à faire cohabiter le C++ traditionnel et le C++ CLR.

5.2 Ecrire l'interface graphique

La première chose que vous allez remarquer c'est que tout le code généré s'écrit dans le fichier .h associé à votre form (code inline). Vous ne devez pas faire pareil. Si le code inline peut être intéressant pour des raisons dont on n'a pas à discuter ici, ce n'est pas dans nos préoccupations, et surtout pas le inline dans la déclaration de la classe : le .h spécifie ce que vous faites, le .cpp dit comment, on évite de mélanger les deux, c'est tout. Il n'y a rien de plus pénible que de débugger, étendre ou réutiliser (et, accessoirement, en ce qui nous concernera corriger) une usine à gaz écrite avec 3000 lignes de codes vomies dans un seul fichier .h.

Donc la question est : "**Comment vous allez faire pour écrire vos traitements ?**". D'abord les **évènements** (clic, déplacement, etc.) sont générés en affichant les propriétés de l'élément sur lequel porte l'évènement, puis en sélectionnant la section "évènements" (symbolisée par un éclair) et en double-cliquant sur l'évènement souhaité. Ceux-ci sont toujours placés dans le .h. La première chose à faire c'est donc de ne pas écrire dans le .h : **vous faites un fichier cpp associé au .h de votre form qui comprendra la fonction générée et vous écrirez son code dedans**.

Un dernier élément important est que quand vous créez une form il la crée dans un "namespace" du nom du projet. Quand vous créez le cpp vous devez donc écrire vos fonctions au sein de ce namespace.

5.3 Les autres classes/fonctions

Un truc assez lourd en ce qui nous concerne (qui est seulement pratique quand les projets deviennent énormes) c'est la présence "d'en-têtes précompilés", le coupable étant le fichier stdafx.h. Sans expliciter le concept, énonçons brièvement quelques règles pour bien les utiliser.

- Tout fichier .cpp doit inclure en premier lieu stdafx.h
- J'insiste la première ligne de tout .cpp DOIT être l'inclusion de stdafx.h, sans ça toute ligne précédente sera ignorée sans qu'une erreur explicite soit retournée

– stdafx.h doit normalement contenir les gros includes. En ce qui vous concerne ça veut dire rien. Vous pouvez quand même y mettre ceux qui seront utilisés par tout le monde comme string, ou d'autres éléments de la stl mais c'est totalement facultatif.

Ensuite les objets. Il y a une différence entre tous les objets manipulés par le CLR et les objets classiques : les objets manipulés par le CLR sont tous (sauf types de base mais string y compris) des pointeurs postfixés par ^. Une chaîne de caractères manipulée par le CLR sera donc notée **String^** et sera en fait un pointeur sur une chaîne. Ceci implique que pour accéder aux propriétés et aux méthodes d'un objet vous devrez utiliser l'opérateur "->".

Les bibliothèques fournies (*"DataBaseAccess"* et *"Intefacer"*) sont abondamment commentées et devraient vous permettre de comprendre l'utilisation des variables CLR. De toute façon les bibliothèques en question ont été écrites pour vous permettre d'éviter au maximum la manipulation de ces variables.

5.4 Exemple utilisant ODBC / VC++ .net

Voici un exemple concret : vous voulez remplir une liste déroulante avec les valeurs résultant de la requête "**select nomprod from produits**", une autre avec les valeurs résultant de l'appel de la **procédure stockée "getCli"** avec le paramètre nomville de type texte valant "Lyon" puis ajouter une commande avec la requête "**insert into commandes (nomClient,nomProd) values ([nom du client],[nom du produit])**".

On place sur la form deux composants de types "**combobox**", pour selectionner le nom du client et le nom du produit nommés respectivement **nomCliCBox** et **nomProdCbox**. On place ensuite **un bouton** pour valider la commande nommé **commandBtn**.

La première chose qu'on veut faire c'est remplir les deux listes déroulantes à l'ouverture de la form. On affiche donc les évènements de la form et on double clique sur l'évènement **Load**, ceci génère une fonction du type suivant dans le .h :

```
private: System::Void Form1_Load(System::Object^  sender,
System::EventArgs^  e) {}
```

On remplace les accolades par un point virgule, puis on crée un .cpp du même nom que le .h qu'on remplit comme il suit :

```
#include "stdafx.h"
#include <string>
#include <list>
#include "Form1.h"
#include "DataBaseAccess.h"
#include "Interfacer.h"

using namespace System;
using namespace System::Windows::Forms;
using namespace System::Data::Odbc;
using namespace std;

namespace NomProj
{
  System::Void Form1::Form1_Load(System::Object^  sender,
    System::EventArgs^  e)
{
//Récupération des données de la première combobox
QueryResult nomProduits = DataBaseAccess::executeQuery(
```

```
      "dsnName",
      "select nomprod from produits");

//Récupération des données de la seconde combobox,
    //constitution des paramètres
list<SqlParameter> parameters;
SqlParameter parameter;
parameter.parameterName = "@nomville";
parameter.parameterType = OdbcType::VarChar;
parameter.parameterValue = "Lyon";
parameters.push_back(parameter);
QueryResult nomClients = DataBaseAccess::executeStoredProc(
      "dsnName",
      "getCli",
      parameters);

//On remplit deux listes de string
    //avec les deux séries de données
list<string> listeProd;
list<list<string>>::iterator prodIter =
      nomProduits.resultSet.begin();
while(prodIter!=nomProduits.resultSet.end())
{
listeProd.push_back(*(prodIter->begin()));
prodIter++;
}

list<string> listeCli;
list<list<string>>::iterator cliIter =
      nomClients.resultSet.begin();
while(prodIter!=nomClients.resultSet.end())
{
listeCli.push_back(*(cliIter->begin()));
cliIter++;
}

//On remplit les deux combobox
nomProdCbox->Items->AddRange(
      Interfacer::getCLRArray(listeProd,listeProd.size())) ;
nomCliCbox->Items->AddRange(
      Interfacer::getCLRArray(listeCli,listeCli.size())) ;
}
}
```

Ceci étant fait on peut faire l'insert sur le bouton. On crée donc de la même façon que précédemment l'évènement **Click** du bouton et de la même façon, on met le code dans le .cpp :

```
System::Void Form1::commandBtn_Click(System::Object^  sender,
  System::EventArgs^  e)
{
//On récupère le texte des deux combobox
String^ cliName = nomCliCbox->SelectedItem->ToString();
String^ prodName = nomProdCbox->SelectedItem->ToString();
```

```
//On utilise ce texte pour compléter l'insert
string sqlQuery = "insert into commandes (nomClient,nomProd)" +
    "  values "(" + Interfacer::toSTLString(cliName) + "," +
    Interfacer::toSTLString(prodName) + ")";

//On exécute l'insert
DataBaseAccess::executeNonQuery("dsnName",sqlQuery);
}
```