Module SIN221
Rendre un système communicant
Analyse UML et algorithmie

Objectifs :

- Modifier des diagrammes UML suite à la modification du cahier des charges.
- Caractériser et valider une classe en C++.

Prérequis :

- langage HTML
- UML
- module SIN21

Documents/ressources nécessaires :

- le projet console C++ Builder de départ donné dans le fichier **SIN22_projet_console_de_depart_pour_stagiaires.zip**

1 Modification du cahier des charges

Aujourd'hui les serveurs web sont implantés dans de nombreux produits industriels. C'est devenu un standard de fait car il est facile de trouver un navigateur pour l'affichage. La modification de notre cahier des charges consiste donc :

- à pouvoir afficher les données présentes dans l'IHM de départ vers une page Web de manière automatique.
- à pouvoir activer ou désactiver la fonctionnalité à la demande.

Le travail de modification suivra les étapes suivantes :

- modification des diagrammes UML utilisés dans le module SIN21 (diagramme de cas d'utilisation, diagramme de séquence, diagramme de classes (imposé celui-là) et diagramme de déploiement).
- Mise au point de la nouvelle classe IHM_WEB (l'algorithme de la méthode **GenererPageWeb**).
- Intégration et validation.

2 Intégration des nouvelles contraintes dans les diagrammes UML.

La modification du cahier des charges n'entraine pas la réécriture complète des diagrammes UML. Au contraire, UML est aussi là pour arriver à mesurer l'impact des changements dans le code du logiciel.

Indications préparatoires :

- Le projet original de la station sous *Visual Paradigm* ne présente que 4 diagrammes. Il est bon de rappeler ici que même si UML 2 propose 13 diagrammes, ils ne sont pas tous à implémenter ! UML propose des outils pour modéliser un système/logiciel le mieux possible. Ici avec 4 diagrammes nous avons largement fait le tour.
- Parmi les 4 diagrammes, le diagramme de classe sera peut-être le plus difficile à comprendre, c'est pour cela qu'il est fourni. Celui-ci représente la structure logicielle. La création d'une page web pour l'affichage des données météo va ainsi avoir pour conséquence la mise au point d'une classe à part qui aura pour rôle la création de cette page. Comme déjà dit, elle se nommera IHM_WEB.

Commentaire [B1]: Ajouter des vidéos ?

Travail à faire

Après avoir bien compris les modifications, vous avez à les traduire sur les diagrammes UML de la manière suivante :

- *Les fonctionnalités décrites par le diagramme de cas d'utilisation sont quasiment identiques sauf que maintenant la consultation devra aussi se faire par navigateur web (indication : il faut penser au lien de spécialisation dans ce cas).*
- *Le diagramme de séquence montre un serveur web en plus. On complètera le diagramme de séquence pour faire apparaitre la ou les interactions avec le logiciel.*
- *Le diagramme de déploiement vous montre la présence de quatre machines distinctes. Vous devez compléter les liens.*

Vous utiliserez le fichier SIN22_OREGON_WEB.vpp pour compléter les diagrammes (à ouvrir avec Visual Paraadigm). Des notes y ont été insérées pour vous guider. Le diagramme de classes vous est donné. Il fait apparaitre une nouvelle classe IHM_WEB dont le travail est de générer une page web à jour des données météo.

3 Algorithme de la méthode GenererPageWeb

La déclaration quasi-complète de la classe IHM_WEB est la suivante :

```
class IHM_WEB {
	private :
		char chemin_enregistrement_page_web[1000] ;
	public :
		IHM_WEB(char* pChemin) //Constructeur
		{
			strcpy(chemin_enregistrement_page_web, pChemin);
		}
		bool GenererPageWeb(int temperature, int point_de_rosee, int humidite,
                                        int etat_meteo, int pression, bool etat_batterie )
		{
                        //CODE DE LA MÉTHODE A COMPLÉTER (VOIR LA SUITE)
		}
} ;
```

Remarque : L'implémentation du constructeur (première méthode appelée automatiquement lors de la création de l'objet et portant le même nom que la classe) ne fait pas partie du travail.

La méthode **GenererPageWeb** doit permettre de créer un fichier html contenant les données à afficher, c'est pour cela que cette méthode prend en paramètre toutes les informations à afficher. Le pseudo–code de la méthode vous est donné ci-dessous :

Méthode GenererPageWeb

Déclarations ***(toutes les variables sont passées en paramètre)***

Temperature : entier
Pression : entier
Point_de_rosee : entier
Etat_meteo : entier
Humidite : entier
Etat_batterie : booléen

Valeur de retour : TRUE (si création fichier ok) ou FALSE

Début

SI Ouvrir/Créer fichier html OK
ALORS
Ecrire entête de page HTML
Ecrire "TEMPERATURE : " puis *Temperature*
Ecrire "PRESSION : " puis *Pression*
Ecrire "HUMIDITE : " puis *Humidite*
Ecrire "POINT DE ROSEE : " puis *Point_de_rosee*
SI *Etat_batterie* vaut TRUE
ALORS
Ecrire "Batterie OK"
SINON
Ecrire "Problème avec la batterie"
FinSI
SELON la variable *Etat_meteo*
CAS 12 : Ecrire "Météo ensoleillée"
CAS 6 : Ecrire "Ciel voilé"
CAS 2 : Ecrire "Couvert"
CAS 3 : Ecrire "pluvieux"
DEFAUT : Ecrire "indéterminé"
FinSELON
Ecrire pied de page HTML
Fermer fichier
Renvoyer TRUE //->signifie que le fichier a été créé
SINON
Renvoyer FALSE //signifie que le fichier n'a pas été créé
FinSI
FIN

Le code de retour de la méthode est un booléen signifiant que la page web a été générée ou pas.

Travail à faire

- *A l'aide de l'algorithme ci-dessus et des éléments qui vous sont donnés en annexe 1, complétez le code de la méthode* ***GenererPageWeb()*** *dans le* ***fichier ihm_web.cpp*** *de votre projet. Votre code doit écrire une syntaxe html dans un fichier ! Vous devez donc écrire des lignes de code html en y mélangeant vos variables (regardez bien le code de départ qui vous est donné comme exemple dans le projet fourni !). Vous testerez dans un projet console (qui n'affiche qu'une fenêtre noire !) votre solution. Pour cela vous utiliserez le projet console fourni que vous n'aurez qu'à compléter (voir annexe 2 pour la structure du projet C++ Builder).*

- ***rem :*** *Lorsque vous lancerez le programme final, vous devrez aller voir le résultat dans le répertoire* ***Debug\Win32*** *de votre projet console. Vous devrez y trouver le fichier html.*

ANNEXE – Ecriture dans un fichier en C++

L'utilisation des fichiers est en général assez simple. En C++ cela se fait en utilisant des objets de la bibliothèque standard. Ici nous ne traiterons que ce dont nous avons besoin, c'est-à-dire de l'écriture dans un fichier, la lecture ne faisant pas partie du projet. Mais les principes sont les mêmes.

Ci-dessous, un exemple de code que je vais commenter pour explications :

```
#include <ofstream>
#include <iostream>
using namespace std;

int main ()
{
	int var = 10;
	ofstream monFichier ;
	monFichier.open("essai.html") ; //ouvre un fichier à l'endroit de l'exécutable
	if monFichier.is_open() { //on vérifie que le fichier est bien ouvert !
		cout << "Fichier ouvert…" << endl; //ligne non obligatoire
		monFichier << "une phrase qui sera écrite dans le fichier" ;
		monFichier << "On peut aussi écrire la valeur d'une variable : " << var;
		monFichier << "et puis encore une autre phrase...";
		monFichier.close() ;
	}
	else
	{
		cout << "Probleme à l'ouverture du fichier !" << endl;
	}
}
```

- **`ofstream monFichier`** : cette ligne permet de déclarer un objet *monFichier* de type *ofstream*.
- **`monFichier.open("essai.html")`** : cette méthode permet tout simplement d'ouvrir le fichier. Il existe des options mais qui ne sont pas utiles pour l'instant. Par défaut, si le fichier existe il est écrasé sinon il est créé.
- **`l'opérateur <<`** : il permet de diriger ce qui est à sa droite vers le fichier ouvert précédemment.
- **`monFichier.close()`** : évidemment, il ferme le fichier.

rem : Les lignes contenant les **cout** permettent d'afficher à l'écran le texte associé. Dans un programme avec une IHM graphique elles doivent disparaitre.

Annexe 2 – Programmation modulaire

Lors du développement d'un logiciel, il convient d'organiser son travail en modules. L'avantage d'une telle démarche est avant tout de pouvoir s'y retrouver. C'est un point crucial dans la qualité du développement. Cela veut dire que vous aurez :

- un fichier principal contenant la fonction **main()** et la ligne principal de votre logiciel.
- un fichier **.cpp** contenant le code des méthodes d'une classe.
- un fichier **.h** contenant le prototype de la classe.

Ainsi, pour notre projet nous avons un fichier **ihm_web.cpp** et un fichier **ihm_web.h**.

Quelques explications sur la syntaxe utilisée dans les différents fichiers en prenant une classe générique appelée **maClasse**.

Fichier de déclaration : *maClasse.h*

Ce fichier sert uniquement à donner le squelette de la classe.

```
class maClasse {

	private :
		// les données d'une classe sont privées en général.
		int val1;
		int val2;

	public :
		//les méthodes sont publiques en général
		maClasse(); //constructeur -> méthode appelée automatiquement à la création de l'objet.
		void Methode1 (char param);  //-> ici on donne le prototype de la méthode et c'est tout.
		int Methode2 (int param); //idem
};
```

Fichier d'implémentation : *maClasse.cpp*

Ce fichier permet d'écrire le code des méthodes. Il y a une syntaxe particulière permettant de le faire. Elle a l'allure suivante :

<Type de retour> <Nom de la Classe>::<Nom de la méthode> (<PARAMETRES>) {

//CODE DE LA METHODE

}

```
#include "maClasse.h" //obligation d'insérer le fichier de déclaration

maClasse::maClasse() {
	//CODE DU CONSTRUCTEUR
}

void maClasse::Methode1(char param) {
	//CODE DE METHODE1
}

int maClasse::Methode2(int param) {
	//CODE DE METHODE2
}
```

Utilisation des fichiers : *main.cpp*

Il faut maintenant instancier un objet et l'utiliser.

```
#include "maClasse.h"  //obliger de déclarer la classe que l'on va utiliser.

int main ()
{
	//déclarations de variables si nécessaire
	char var;
	int resultat;

	maClasse unObjet;            //création d'un objet. On appelle cette opération une instanciation
	                             // on instancie un objet de la classe maClasse
	unObjet.Methode1(var);       //appel de la méthode 1
	resultat=unObjet.Methode2(7); //appel de la méthode 2
	//etc
	return 0;
}
```