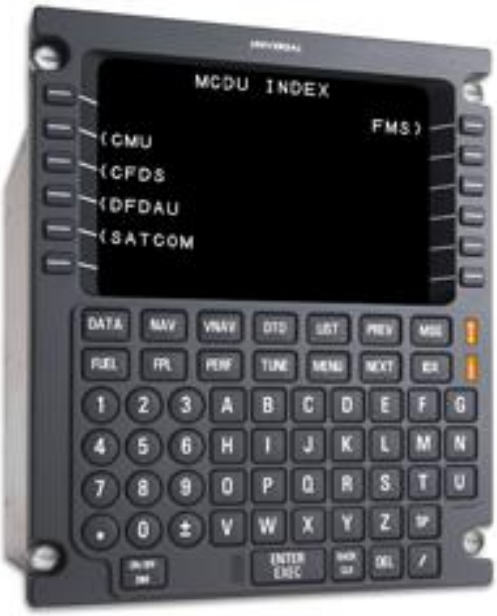
UNIVERSITE DE LA MEDITERRANEE
AIX-MARSEILLE II
I.U.T. d'Aix-en-Provence
Département Informatique

RAPPORT DE STAGE

VSM
Interface Homme Machine : Le MCDU

ANNEXES



Maximilien MOUSSALLI

Enseignant Responsable : Vincent RISCH
Enseignant Lecteur : Marc LAPORTE

JUIN 2008

SOMMAIRE

Annexe 1 : Structure des pages version 1.0

Structure des pages :

- Une classe mère appelée « **PageAbstr** » contenant :
  - Une liste de pointeurs de « **PageAbstr** » (un pour chaque bouton de 1L jusqu'à 6R plus les touches NEXT et PREV). Pour permettre de définir les liens des pages accessibles correspondant aux boutons.
  - Des fonctions comme :
    - **GetPage** qui renvoie le pointeur de « **PageAbstr** » correspondant à une touche passée en paramètre.
    - **InitLien** (fonction virtuelle) qui permet à la classe fille l'attribution de ses liens avec les autres pages, grâce à une liste de pointeurs de « **PageAbstr** » passée en paramètre, qui contient l'ensemble des pages du « MCDU ».
    - **WriteBuffer** (fonction virtuelle) qui écrit le contenu de la page correspondante dans un « **buffer** » passé en paramètre.

- Une classe fille qui dérive de « **PageAbstr** », appelée « **PageNOMDELAPAGE** » pour chaque page du MCDU à créer afin de gérer plus facilement les nombreux cas de traitements spécifiques liés à chaque page.

- Une classe appelée « **CPage** » qui me sert d'interface, c'est-à-dire qui se charge de regrouper toutes les pages du MCDU et de les manipuler facilement. Cette classe contient :
  - La liste des pointeurs de « **PageAbstr** » de l'ensemble des pages du MCDU.
  - Un pointeur de « **PageAbstr** » qui définit la page courante sur laquelle on se trouve actuellement.
  - Des fonctions comme :
    - **InitPages** qui se charge d'initialiser la liste des pointeurs avec les pages correspondantes ainsi que d'initialiser la page courante.
    - **WriteBuffer** qui se contente d'appeler la fonction **WriteBuffer** de la page courante.

Annexe 2 : Structure des pages version 1.1

J'ai dû modifier la classe « **CPage** » pour permettre la gestion des changements de page. J'ai rajouté la fonction **Action** qui prend en paramètre le numéro d'une touche (1L à 6R) et qui grâce à la fonction **GetPage** effectuée sur la page courante, me permet de récupérer le pointeur de l'éventuelle page à laquelle correspond la touche. Si le pointeur récupéré n'est pas nul alors la page courante devient le pointeur récupéré.

Annexe 3 : Structure des pages version 1.2

Les modifications à faire sur les différentes classes pour permettre à l'utilisateur de modifier des variables sont :

- « **PageAbstr** » :
  - Ajouter la fonction **SetAction** (fonction virtuelle) qui prend en paramètre le numéro d'une touche et une chaîne de caractères (que l'utilisateur aura saisi) qui permettra à une classe fille de gérer une action en fonction du bouton pressé et de la ligne saisie par l'utilisateur.
- « **CPage** » :
  - L'ajout d'une chaîne de caractères en donnée membre qui servira à stocker la chaîne de caractères que l'utilisateur aura saisie.
  - L'ajout d'une fonction **SetMessage** qui remplace la valeur de la chaîne de caractères en donnée membre par celle passée en paramètre de la fonction.
  - Dans la fonction **WriteBuffer** :
    - L'ajout d'une partie qui s'occupe d'afficher dans le « scratch pad », la chaîne de caractères que l'utilisateur aura saisie.
  - Dans la fonction **Action** :
    - Avant les nouvelles modifications, pour une touche donnée, cette fonction regardait s'il existait une page grâce à la fonction **GetPage** de la page courante, et changeait la page courante si une page existait. Maintenant si la page n'existe pas, alors on appelle la fonction **SetAction** de la page courante en lui passant en paramètre le numéro du bouton pressé, ainsi que la donnée membre qui contient la chaîne de caractères saisie par l'utilisateur. Cette fonction se chargera alors de traiter les modifications propres à la page courante.

Annexe 4 : Quelques Fonctions intéressantes

```
//Fonction qui transforme un string en unsigned
//Ne gère pas les signes ni les virgules.
//renvoie vrai si l'action a reussi, faux sinon.
static bool StrToUInt (const std::string & Str, unsigned & Num)
{
	if (!IsDigit(Str)) return false;

	Num = 0;
	for (unsigned i = 0; i < Str.size(); ++i)
		Num = (Num * 10) + (Str [i] - '0');

	return true;
}

//Fonction qui transforme un unsigned en string sous forme
//decimale
static void UIntToStrDec (const unsigned Num, std::string & Str)
{
	Str.clear();

	char C = (Num % 10) + '0';

	Str.insert(Str.begin(), C);

	unsigned mod = 100;
	unsigned div = 10;

	while (Num / div)
	{
		C = ((Num % mod) / div) + '0';
		Str.insert(Str.begin(), C);
		mod *= 10;
		div *= 10;
	}
}

//Fonction qui transforme un unsigned en string en représentation
//binaire avec « Taille » comme nombre de chiffres pris en compte
static void UIntToStrBin (const unsigned Num, std::string & Str,
                          const unsigned Taille)
{
	Str.clear();
	Str.resize(Taille);

	for (std::size_t i = Str.size(), j = 0; i--; ++j)
		Str[i] = (Num & (unsigned(1) << j))? '1':'0';
}
```

Annexe 5 : Structure des pages version 1.3

Modifications apportées pour rendre les liens d'une page dynamique :

La fonction **GetPage** de la classe « **PageAbstr** » devient virtuelle afin de permettre, à une classe fille, de rendre ses liens dynamiques en fonction des critères qui lui conviendront le mieux.

Dans la classe « **PageAbstr** » la fonction **GetPage** reste la même qu'avant modification, afin de permettre d'équiper les pages filles plus classiques, de la gestion normale des liens vers leurs pages.

Annexe 6 : Structure des pages version 2.0

Modifications apportées pour permettre de récupérer le contenu d'une autre page :

Dans toutes les classes des pages filles déjà existantes, la fonction virtuelle « **WriteBuffer »** devient la fonction virtuelle **GetBuffer** qui ne s'occupe plus, d'écrire dans le « **buffer** » la page mais de renvoyer seulement son contenu.

C'est ensuite dans la fonction **WriteBuffer** (qui n'est plus virtuelle) de la classe « **PageAbstr** » que l'écriture de la page dans le « **buffer** » est effectuée en utilisant la fonction **GetBuffer** pour récupérer le contenu de la page.

Annexe 7 : Structure des pages version 3.0

Modifications apportées pour rendre la structure des pages plus souple et plus intuitive :

Remplacement de la fonction **GetBuffer** dans toutes les pages par trois fonctions, une pour le titre, une pour le corps et une pour le pied de page, que j'appellerai **GetHeader**, **GetBody**, et **GetFooter** qui renverront chacune leurs valeurs propres. Cette structure est surtout plus pratique pour les pages contenant, elles mêmes, plusieurs pages. Permettant la répétition d'un titre et d'un pied de page variable automatiquement.

Annexe 8 : le fichier PageAbstr.h

```
class PageAbstr
{
	protected :
		//definit si le numero de page et du nombre de pages doit
		//être affiché.
		bool	m_NbPagesVisible;
		//contient le nombre total de pages.
		unsigned m_NbPages;
		//contient le numéro de la page courante.
		unsigned m_NumPage;

		//contient la liste des liens des pages.
		std::vector <PageAbstr *> m_Pages;

	public:
		PageAbstr(void);
		virtual ~PageAbstr(void);

		//initialise les liens avec les autres pages contenues
		//dans la liste
		//de l'ensemble des pages du mcdu passé en paramètre.
		virtual void  InitLien
			(const std::vector <PageAbstr *> & Pages);

		//contient les commandes à effectuer lors de
		//l'appel de la page.
		virtual void LoadPage ();

		//renvoie l'en-tête de la page.
		virtual std::vector <std::string> GetHeader ();
		//renvoie le corps de la page.
		virtual std::vector <std::string> GetBody   ();
		//renvoie le pied de page de la page.
		virtual std::vector <std::string> GetFooter ();

		//effectue les actions de la page en fonction
		//de la touche et du message
		//passés en parametre.
		//cette fonction renvoie un statut de message d'erreur
		virtual nsPages::Message SetAction
			(const Touche t, std::string & Str);

		//renvoie le lien de la page correspondante à la touche
		//passée en paramètre
		virtual PageAbstr * GetPage (const Touche t) const;

		//Ecrit dans le buffer le contenu de la page.
		void WriteBuffer (text_buffer_t & text);

};
```

Annexe 9 : le fichier CPage.h

```
class CPage
{
	protected :
		//pour stocker la liste de l'ensemble des pages du mcdu.
		std::vector <PageAbstr *>	m_Pages;
		//contient le pointeur de la page courante.
		PageAbstr *			m_PageCourante;
		//contient la saisie de l'utilisateur.
		std::string			m_Message;
		//contient le message d'erreur qu'une page peut renvoyer.
		Message				m_MsgErreur;

	public:
		CPage(void);
		virtual ~CPage(void);

		//Initialise l'ensemble des pages et les met
		//dans « m_Pages ».
		void InitPages ();

		//Ecrit dans le buffer le contenu de la page courante.
		void WriteBuffer (text_buffer_t & text);

		//Effectue les actions correspondantes à la touche
		//passée en paramètre et à la page courante.
		//un message d'erreur peut être écrit dans
		//« m_MsgErreur ».
		void Action (const Touche t);

		//Pour modifier « m_Message ».
		void SetMessage (std::string Msg);

};
```