SMILE :-)

Conception module affichage

SOMMAIRE

1. Introduction

Dans le cadre du projet OPALE, et plus particulièrement en ce qui concerne le projet SMILE, l'ensemble des informations concernant la conception se trouve dans le présent document. Cependant pour des motivations de clartés et de facilité d'accès, les documents de conceptions, tout commes les dossiers de spécifications sont répartis en trois documents distinct... Vous avez entre les mains le dossier de conception concernant **le module d'affichage**.

Ce document a été réalisé avec OpenOffice, et les diagrammes avec Dia. Les sources sont disponibles sur le serveur d'IMERIR ou surr notre site internet. Les numéros entre crochet font référence à un lien que l'on trouvera en dernière page.

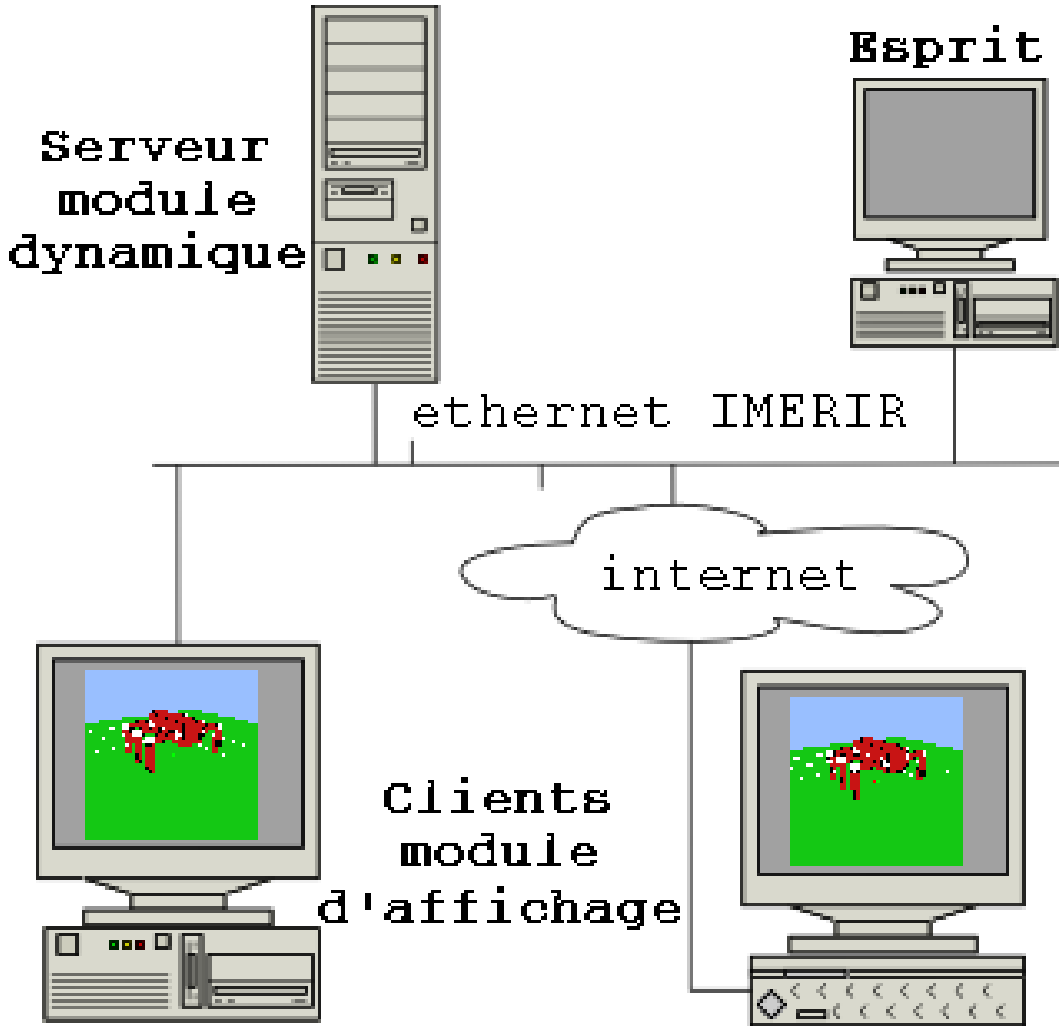
Vous trouverez dans ce document la description de l'architecture du système à différents niveaux.

La première partie décrit son contexte, sa composition, et donne un diagramme de classe general.

La deuxième partie présente un diagramme de classe detaillé.

2. Conception préliminaire

1. Le système dans son contexte



2. Les composants du système

Le système que nous développons cette année se décompose en trois parties :

- un module Opale
- un module Dynamique
- un module Affichage

Le schéma suivant illustre la répartition de chaque module et les relations qui les font communiquer.

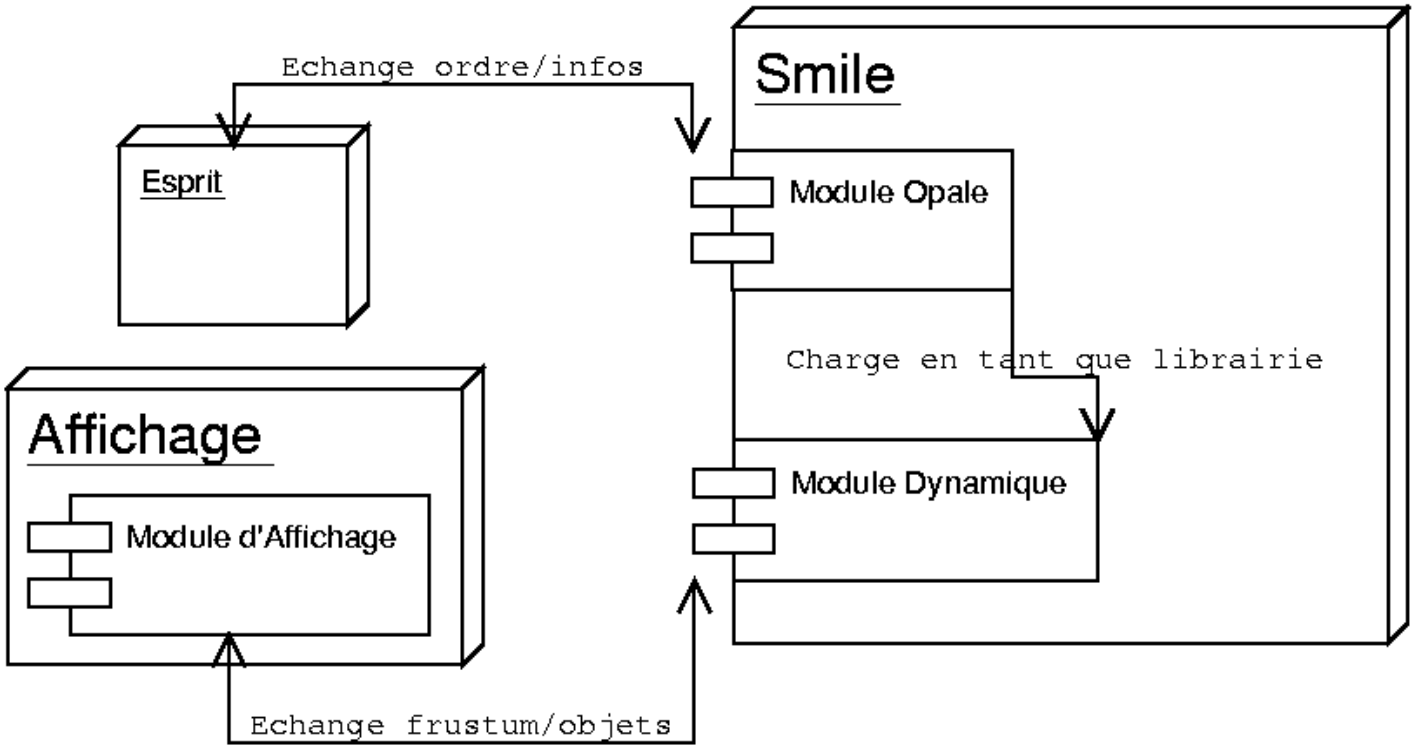


Illustration 1 Schéma des composants du système

3. Diagramme de classe du module d'affichage dans le système

Ce diagramme de classe suit la norme UML. Cependant nous rappelons la légende ci-dessous.

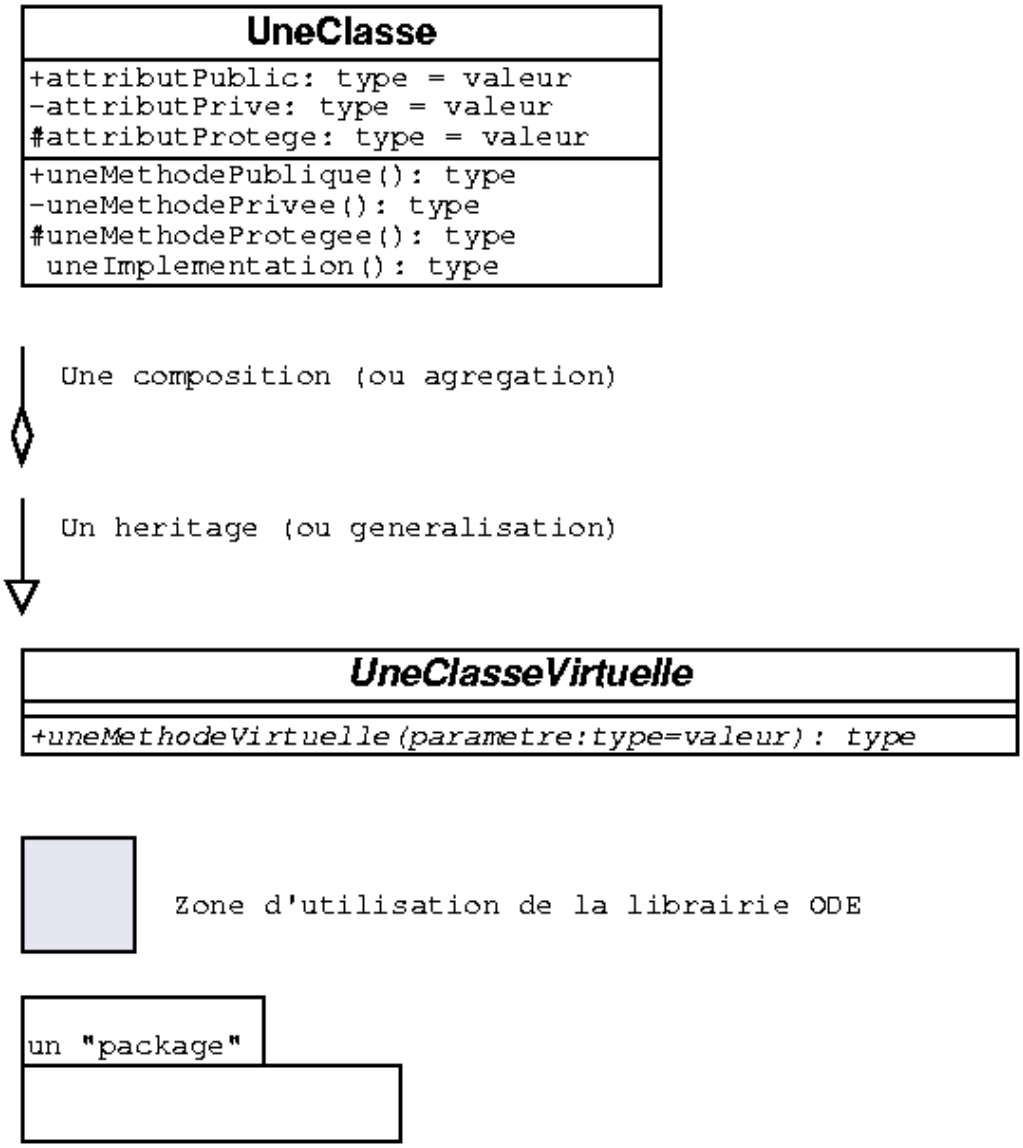


Illustration 2 Légende des diagrammes de classe

Voir le diagramme en annexe

3. Conception détaillée

1. Environement de développement et methode d'affichage

A. Environement de développement

Description de l'environnement de développement par ressources :

Systèmes d'exploitation:

Windows NT/2000/XP.

GNU/Linux.

Logiciels de développement:

Microsoft Visual C++

gcc

Bibliothèques :

ODE / OpenGL / Glut ...

Matériels :

Les contraintes matérielles se situent sur les points ci-dessous qui seront très sollicités durant la simultation :

Reseau Ethernet / Internet (celui d'IMERIR)

Cartes Graphiques OpenGL (pour le module graphique)

Ram (pour le serveur qui fera tourner le module dynamique)

Fréquence processeur élevé (pour le serveur qui fera tourner le module dynamique)

Humain :

l'équipe projet (5 personnes).

B. Methode d'affichage

Le fonctionnement du client graphique se passe ainsi : la caméra fait une requête pour connaitre les objets qui sont dans son champ de vision. Une liste de primitives lui est retournée. C'est cette liste qui sera alors affichée à l'écran tant que la caméra ne fera pas une autre demande du contenu de son champ de vision. Cette méthode sacrifie au rafraichissement des données une rapidité de calcul car le serveur n'aura pas à envoyer tout le temps à tous les clients graphiques le contenu de leur champ de vision. Bien que cette dernière méthode soit préférable quand à la validité des objets affichés, la somme de calculs effectués dans ce genre de traitement et celle des autres traitements en parallèles nous fait chosir la solution de l'affichage sur requête de la caméra. Cette requête pourra être lancée notamment quand la caméra bougera.

2. Description par classe

Nous concevons ce module dans une optique de codage orienté objet. Cependant, le codage sous OpenGL est généralement fait de façon procédurale, et nous n'excluons pas que ce module soit codé en procédurale.

Classe Fenetre :

Les constructeurs .

Le constructeur de la fenêtre est en fait une fonction de type CreateWindows (pour OpenGL) qui permet de créer une fenêtre dans laquelle OpenGL viendra dessiner. A cette fonction nous lui passerons les paramètres de taille et de détail voulus.

Les accesseurs .

Les autres methodes :

```
int init(unsigned short longueur, unsigned short hauteur, unsigned char detail)
```

Première opération à faire avant d'afficher la scène.

Création de la fenêtre.

```
{
Appel de la fonction de création de la fenêtre (CreateGLWindows() )
Appel de la fonction d'initialisation (InitGL() )
Choix du type de forme des objets (Flat/Smooth)
Couleur de fond
Buffer de Profondeur
Gestion du clavier et de la souris
}
```

```
void setCamera(double oeil[3], double cible[3],double fov,double profondeur)
```

Définition de la caméra : champ de vue, profondeur, point de départ (oeil), er(cible)point regard

```
{
Création de la caméra en utilisant les arguments
}
```

```
void rafraichir(liste objet)
```

Redessiner tout les objets de la scène pour la mettre à jour

```
{
Creation d'un objet obj
obj = tous les objets en collision avec le frustum(Interface.getObjetIn(frustum))
pour tous ces objets
obj.Dessine() // on dessine chaque objet
}
```

Classe Objet :

Note : il faut que cette classe soit virtuelle.

Les constructeurs .

Les accesseurs :

Ces methodes sont virtuelles pures. Elles définissent l'interface que devrons impérativement avoir les objets finaux.

Les autres methodes :

La liste d'objets sera triée. Nous connaitrons les types de chaque objets et nous les passerons ensuite à la méthode Dessine générale qui, elle, appellera la méthode de dessin qu'il convient.

```
void Dessine(type objet)
                                    Cette méthode est virtuelle, chaque objet se dessinera différemment.
Selon (type)
 cas Boite:
  DessinerBoite(objet*)
 cas Cone:
  DessinerCone(objet*)
 cas Pyramide:
  DessinerPyramide(objet*)
...
```

Classes Plan, Boite, Sphere, Ccylindre et Charnière :

Les constructeurs .

Les accesseurs :

Les autres methodes :

L'objet passé en paramétre est un pointeur sur une structure contenant les informations nécessaire à dessiner un objet. Nous aurons donc le type qui servira à dessiner le bon objet, mais aussi le centre de gravité pour placer l'objet, les longueurs (ou rayon...) pour avoir sa taille, sa couleur, etc.

La structure objet est standard, donc tous les champs ne seront pas forcément renseignés.

```
void Dessine(objet*)
```

Dessine un objet de n'importe quelle nature. Cette méthode est surchargée, propre à chaque type d'objet

```
{
Effacer le contenu de l'écran
Selon les paramètres (longueur, hauteur, diamètre, angle de buter)
Construire l'objet
}
```

Classe Interface:

Les constructeurs .

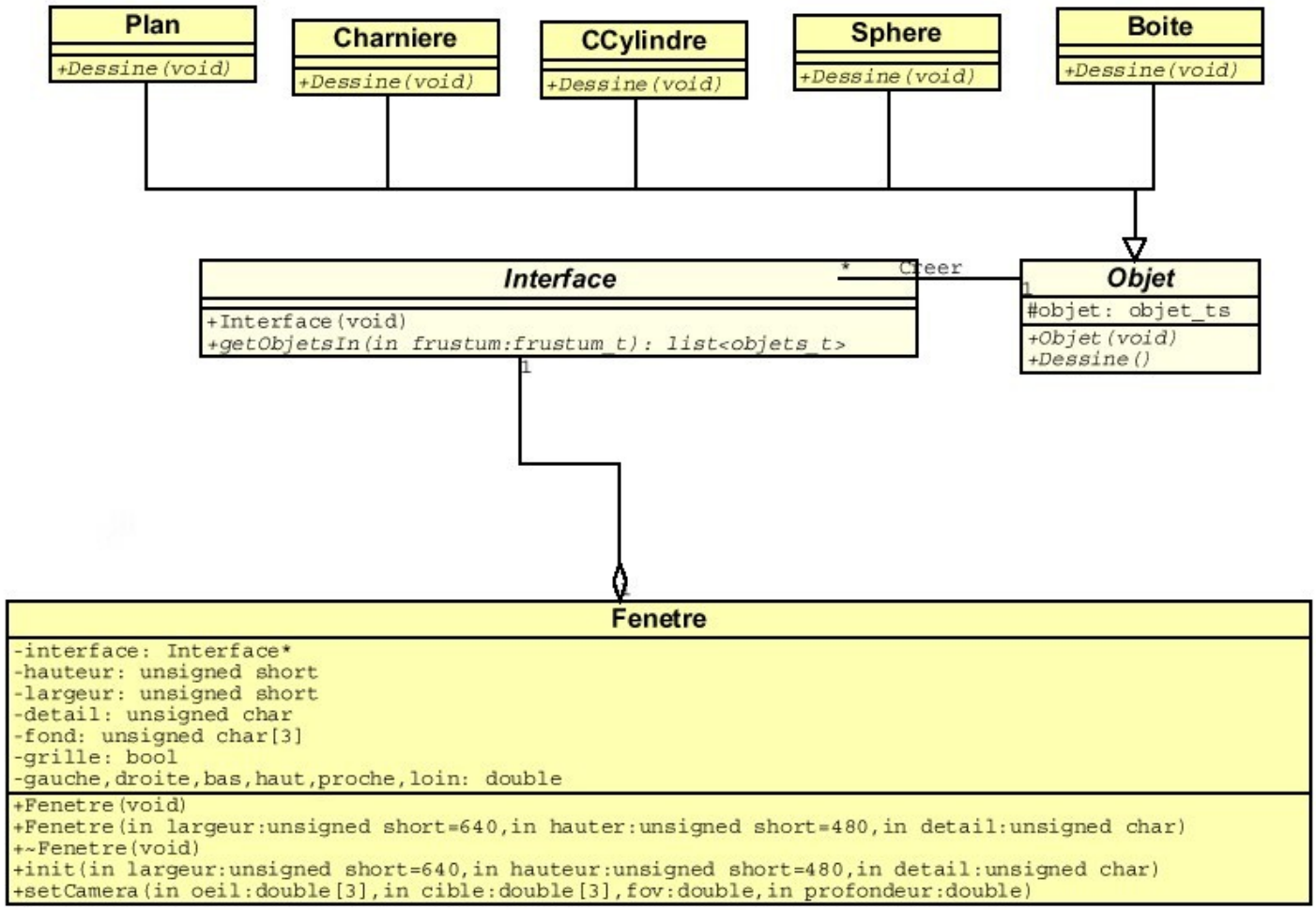
Les accesseurs .

Les autres methodes :

```
list_objet GetObjetsIn(frustum monfrustum)
                                    Récupère  la liste des objets qui sont dans le champ de vision de la caméra
{
Appel de la fonction vers la Classe DialogThread
Récupère la liste
Appel la fonction Rafraichir() en passant la liste des objets en paramètres
}
```

4.Annexes :

Diagramme de classe :



liens :

http://www.lysator.liu.se/~alla/dia/

chemin/vers/le/cahierdescharges

http://www.CM-Labs.com

http://opende.sourceforge.net/

assarssonXXoptimized.pdf

http://opende.sourceforge.net/ode-latest-userguide.html#ref54