INTERFACE GRAPHIQUE VIRTUELLE POUR LES PROGRAMMES PARALLELES.

Pierre MOUKELI
Institut Africain d'Informatique
B.P. 2263 Libreville
Fax: (241) 72 00 11
GABON

Résumé.

Cet article propose un modèle et une démarche de conception d'interfaces graphiques de programmes parallèles. L'interface graphique a jusque là été considérée comme un outil secondaire mais pourtant très utile aux programmes parallèles. Sa conception et sa gestion sont strictement séparées du programme parallèle, alors qu'elle en est la plupart du temps un composante essentielle. Nous proposons de ramener au moins partiellement cette gestion dans le programme parallèle, en utilisant le concept d'interface graphique virtuelle.

Mots clés: Parallélisme, Interface graphique, moniteur.

Introduction.

L'interface d'une application parallèle nécessite un ensemble de fenêtres et un noyau qui les gère, le tout constituant l'interface graphique. Mais il faut également un moniteur de transport bidirectionnel des données entre les processus de l'application parallèle (appelés processus d'application dans la suite) et l'interface graphique; et des serveurs permettant la collecte et l'acheminement des observations produites par les processus d'application. Beaucoup d'efforts ont été déployés par la communauté scientifique pour réaliser de telles interfaces [1][4][5][11].

Le problème majeur que pose la conception des interfaces graphiques, outre leur complexité, est le décalage fantastique existant entre la vitesse de la machine parallèle et celle de la station qui supporte l'interface. En effet, les machines parallèles sont conçues pour gagner en temps de calcul. Les interfaces graphiques sont réalisées avec le souci du confort d'utilisation. Fonctionnant à l'échelle du programmeur, ces interfaces ne sont pas faites pour collaborer efficacement avec la machine parallèle. En effet, tout envoi de message du programmeur à un processus conduit à ralentir considérablement la machine parallèle. Réciproquement l'affichage en temps réel des résultats

d'une exécution conduit à des congestions dans le réseau de communication et à une élimination très sévères des données à visualiser du fait de la lenteur de l'interface graphique. Bref, faire coopérer l'interface graphique avec une machine parallèle conduit fatalement à une dégradation de la vitesse de la machine parallèle, et à un appauvrissement des échanges. On est presque résigné à ne utiliser l'interface graphique qu'en différé. En conséquence, la visualisation différée conduit fatalement à dissocier totalement le programme parallèle de la présentation et de la visualisation des résultats.

L'idée novatrice que nous présentons dans cet article vise à concilier ce qui semble à cette date inconciliable : rendre l'interface graphique accessible en temps réel aux processus, tout en préservant les performances de la machine parallèle; et permettre que la présentation des résultats puisse être programmée directement par les processus utilisateurs. Pour ce faire, nous proposons le concept d'interface graphique virtuelle. Celle-ci consiste pour les processus à avoir localement une image de l'interface réelle, et de ne s'adresser qu'à cette image. Le système se charge d'assurer en ligne la conformité de l'image avec les fenêtres graphiques réelles, en rafraîchissant ces dernières avec les vues successives des fenêtres virtuelles. En particulier, ces vues collectées dans des fichiers peuvent permettre de différer la visualisation, tout en respectant la présentation imposées par les processus. Fonctionnant au niveau même de la machine parallèle, l'interface graphique virtuelle est accessible avec les mêmes niveaux de performance que toutes les autres ressources de la machine. Les processus parallèles sont libérés de lenteur de la vitesse d'affichage de l'écran graphique. De plus, la gestion de l'interface graphique est faite dans le code source des processus, ce qui uniformise la programmation. C'est là des avantages certains sur les interfaces graphiques existantes.

Cet article se compose de trois parties. La première décrit en détail les problèmes liés à la conception des interfaces graphiques. Elle expose aussi nos motivations, et donne quelques hypothèses sur l'environnement de programmation cible. La deuxième partie décrit le modèle et la méthode de conception de l'interface graphique que nous préconisons, s'articulant autour des fenêtres virtuelles. Enfin, la troisième partie donne des indications de mise en oeuvre en s'appuyant sur l'expérience tirée de CDS [8][9]. Les exemples donnés s'inspirent du langage C.

I. Présentation du problème.

1.1. Problème.

Une interface graphique est un outil central dans la programmation parallèle, car c'est l'image visible de l'exécution. Cependant les interfaces existant actuellement posent quelques problèmes.

- Ces interfaces ont souvent évolué avec les langages parallèles vers des **outils offrant des services standard** (e.g. affichage de traces, mesure de performances). De ce fait, le programmeur ne peut les personnaliser pour ses besoins spécifiques. Finalement c'est le programme qui s'adapte à l'interface et pas l'inverse.

- Il est encore d'usage que **le programmeur réalise intégralement son interface graphique** (conception des fenêtres, écriture des fonctions de gestion de ces fenêtres, réalisation du noyau de contrôle de l'interface graphique, écriture des fonctions de communication). Or programmer une interface graphique est une opération fastidieuse (e.g. utilisation de Xwindow).

- **La visualisation est séparée du programme principal**, alors que dans beaucoup de problèmes, elle est une composante de l'application (e.g. mesure de performances, traitement d'images). En fait, il faudrait repenser les langages de programmation parallèle pour y intégrer la gestion de l'interface graphique. Car si l'on considérait l'interface graphique comme une extension du système d'entrée sortie, alors sa programmation serait au moins partiellement intégrée au programme parallèle comme dans les applications transactionnelles.

- **Les interfaces actuellement proposées sont en général statiques, et souvent ne reflètent pas le parallélisme de l'application.** Statiques, elles le sont du fait que les fenêtres sont strictement contrôlées par l'utilisateur. Les processus ne modifient l'état de celles-ci qu'indirectement à travers les données qu'ils collectent. Or donner un minimum de contrôle des fenêtres aux processus d'application permettrait de donner à la visualisation à la fois, un caractère instantané de l'exécution, et un reflet du parallélisme du fait que certains processus peuvent avoir leurs propres représentations graphiques (e.g. fenêtre) qu'ils peuvent modifier librement.

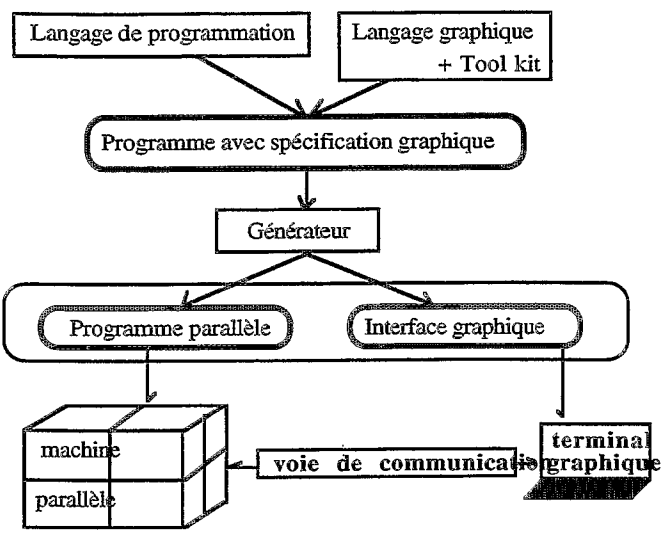
1.2. Objectifs.

Notre objectif est de proposer un modèle et une démarche de conception prenant en compte la gestion de l'interface graphique au niveau du programme

parallèle. En effet, en partie gérée par les processus parallèles, l'interface graphique devient dynamique et reflète le parallélisme de l'application. Une partie de cette interface étant spécifiée par le programmeur, il peut l'adapter en conséquence à ses besoins.

Cela nécessiterait soit l'utilisation d'une bibliothèque de fonctions, soit l'extension du langage parallèle avec des instructions de spécification graphique. Dans le premier cas, la charge de l'utilisateur sera non négligeable du fait de la conception des fenêtres. Mais la mise en oeuvre est facile. Dans le deuxième cas, un générateur se chargera d'extraire cette spécification pour générer l'interface graphique et un programme parallèle comportant des communications avec l'interface graphique. La mise en oeuvre de cette deuxième hypothèse est plus complexe en particulier du fait de la conception du générateur.

Dans les deux cas, le langage de programmation est étendu avec des instructions (ou fonctions) de spécification graphique. Ces instructions seront éventuellement reconnues par un automate qui les extraira du programme source, pour d'une part leur substituer des instructions de communication avec l'interface graphique; et d'autre part, générer un nouveau programme constituant l'interface graphique. Ce scénario est représenté par la figure 1.



-Figure 1- Structure de l'environnement graphique.

Cette solution pose un ensemble de problèmes. Le premier est de savoir comment effectuer la spécification des fenêtres par rapport au programme

parallèle. Le deuxième problème est de gérer la communication entre les processus et l'interface graphique. Le troisième problème est de garantir l'intégrité, la consistance et la fiabilité du système. En effet que se passerait il, si un processus envoie une requête destinée à une fenêtre qui n'existe pas?

1.3. Hypothèses sur l'environnement de programmation.

Dans cet article, nous supposons disposer d'un langage de programmation parallèle, et d'un moniteur interactif permettant la communication entre l'application parallèle et l'interface graphique. Ce moniteur a pour rôle de collecter et de convoyer les informations entre les processus et l'interface graphique, dans un flux bidirectionnel.

Sans être restrictif, nous choisissons le modèle de programmation MIMD, avec un parallélisme explicite. Nous supposons disposer d'un environnement matériel et logiciel permettant de communiquer entre une station de travail graphique et la machine parallèle cible. Nous supposons aussi, que cet environnement permet de concevoir des interfaces graphiques [11] (e.g. Xwindow, Xview, Sunview).

II. Interface graphique virtuelle.

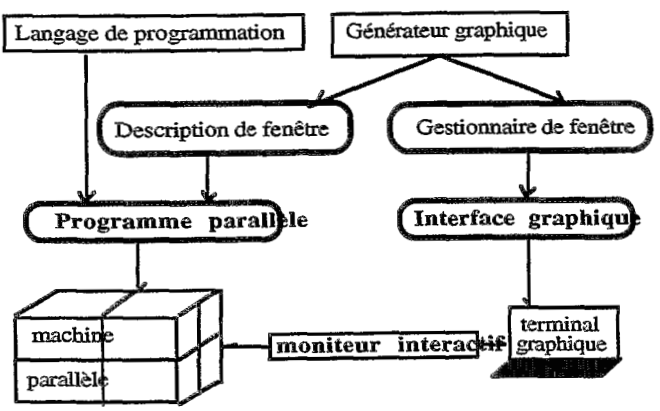
Le problème majeur qui se pose lorsque l'on veut associer des fenêtres à des processus d'une machine parallèle, est l'écart de vitesse considérable entre le terminal graphique (généralement une station de travail) et la machine parallèle. Le même problème s'étant posé entre le micro processeur (trop rapide) et la mémoire centrale (trop lente), on y a répondu en utilisant des mémoires caches. Suivant la même philosophie, afin de palier à cet écart, les processus doivent avoir une image interne de chaque fenêtre, à laquelle ils peuvent accéder plus rapidement que l'écran réel. Cette image sera appelée fenêtre virtuelle, et l'ensemble de ces images constitue l'interface graphique virtuelle. Le système se chargera de maintenir la cohérence entre fenêtre virtuelle et fenêtre réelle. En plus du gain de vitesse, un autre avantage des fenêtres virtuelles est qu'on peut enregistrer des vues pour le play-back; nous montrerons comment dans la suite.

Pour concevoir une interface graphique, on peut procéder manuellement en concevant chaque fenêtre avec ses fonctions de gestion; ou utiliser un générateur capable d'extraire la spécification de l'interface du programme parallèle. Dans le premier cas la charge du programmeur est relativement importante car la conception d'une interface graphique est laborieuse. Dans le second cas, la conception du générateur est relativement complexe.

Nous privilégions une solution intermédiaire, alliant les deux stratégies précédentes. Son principe est le suivant. Les fenêtres sont conçues à l'aide d'un générateur de fenêtres. Le programmeur complète le code généré avec les fonctions de gestion de la fenêtre. Le générateur des fenêtres produit également une spécification qui sera intégrée au programme parallèle. Le programmeur écrit ensuite le programme parallèle dans son langage favori, en y intégrant la spécification des fenêtres et les opérations de manipulation des celles-ci. La figure 2 schématise notre solution.

En plus du générateur de fenêtres, le programmeur dispose de deux autres outils actifs pendant l'exécution du programme et son interface. Le premier est le noyau chargé de gérer toutes les fenêtres. Le second est un ensemble de processus serveurs s'exécutant un sur chaque noeud de la machine parallèle. Leur rôle est de gérer l'accès des processus à l'interface graphique.

Le reste de cette partie détaille cette solution. Nous présentons successivement la spécification, la génération et le fonctionnement du programme parallèle autour de l'interface graphique.



-Figure 2- Environnement de l'interface graphique.

2.1. La spécification.

Cette phase a un double objectif. Le premier est de construire les fenêtres qui constitueront l'interface graphique et de leur associer des fonctions de manipulation. Le deuxième est de spécifier des opérations à effectuer sur ces fenêtres à l'initiative des processus parallèles. Mais avant de décrire ces deux types de spécification, nous allons d'abord exposer la représentation des fenêtres au niveau des processus, et en montrer l'intérêt; ainsi que les opérations qu'ils peuvent effectuer sur ces fenêtres.

2.1.1. Fenêtres virtuelles.

La fenêtre virtuelle est un objet global ou distribué, selon que la machine parallèle cible possède une mémoire partagée ou distribuée, manipulé par les processus d'application. Plus précisément, ce sera une structure de données, qui, pour minimiser l'espace mémoire, ne comportera que les éléments essentiels pour modifier l'état d'une fenêtre(e.g. ligne courante, valeur des champs et leurs attributs comme la couleur, état de chaque champ). Toutes les opérations effectuées par les processus porteront sur ces fenêtres virtuelles; le système se chargera de répercuter ces opérations sur les fenêtres réelles, en temps réel ou en différé. Nous y reviendrons en détail.

2.1.2. Opérations sur les fenêtres virtuelles.

Les opérations effectuées par les processus pour exploiter l'interface graphique ont pour effet d'afficher les données collectées par ceux-ci, recevoir les requêtes du programmeur, et administrer les fenêtres. Ces opérations portent sur les fenêtres virtuelles comme nous l'avons précédemment dit, et le système les répercute sur les fenêtres réelles. Elles peuvent être spécifiées sous forme d'instructions ou de fonctions dans le langage de programmation parallèle; nous retenons cette dernière hypothèse dans la suite pour sa facilité de mise en oeuvre. Pour des raisons de normalisation et de probabilité, ces opérations doivent être pré définies et standard, avec la possibilité pour le programmeur d'en définir des plus complexes à partir de ces opérations de base.

Etant des objets internes à la machine parallèle, les fenêtres peuvent se voir appliquer la plupart des opérations réalisables sur des objets. Mais étant l'image de fenêtres réelles, on doit aussi leur appliquer certaines opérations réalisables sur les fenêtres réelles. Sans être exhaustif, les plus significatives de ces opérations sont :

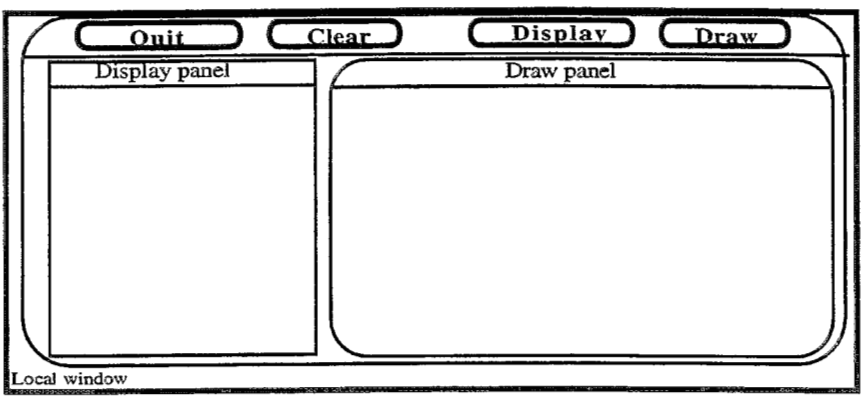
- *Create* : crée une fenêtre virtuelle pour un noeud de la machine parallèle, ou pour un processus. Cette opération donne lieu à l'initialisation d'un objet de type fenêtre virtuelle.
- *Remove* : détruit une fenêtre créée par *Create*.
- *Display* : affiche un message de type donné dans une fenêtre virtuelle.
- *Draw* : dessine un bitmap dans une fenêtre virtuelle.
- *Get_from* : reçoit une requête du programmeur à partir d'une fenêtre virtuelle.
- *Broadcast* : affiche un message dans un ensemble de fenêtres virtuelles.

2.1.3. Conception des fenêtres.

La conception d'une fenêtre par programme étant en général très lourde, elle devra se faire à travers un outil dédié, logiciel ou graphique (e.g. GUIDE sous Xview). La fenêtre étant ensuite manipulée comme un objet sur la station graphique, cet outil devra également générer les fonctions de gestion. Par exemple, à un bouton sera associée une fonction appelée chaque fois qu'il sera activé par l'utilisateur. Le programmeur devra éventuellement modifier ces fonctions pour les personnaliser.

On peut fondamentalement considérer trois types de fenêtres. Les fenêtres globales, une par type de service (e.g: fenêtre de contrôle, tableau de bord), sont accessibles à tous les processus. Les fenêtre locales, sont associées à chaque noeud; leur usage est limité aux processus s'exécutant sur ce noeud. Et enfin, les fenêtres individuelles permettent à certains processus d'effectuer des opérations personnalisées.

Exemple.



-Figure 3- Exemple de fenêtre locale

2.1.4. Programme parallèle.

Nous avons dit que les fenêtres réelles et le programme parallèle sont construits séparément. Le programme parallèle comportera en plus des instructions classiques, la gestion des fenêtres virtuelles à l'aide des opérations adéquates disponibles dans une bibliothèque comme dans [12]. La spécification des fenêtres virtuelles sera inclue dans le code source des processus, comme illustré par l'exemple suivant.

Exemple:

```
#include "virtualwin1.h"          /* virtual window win1 specification file */
Bitmap   mat[150][150] ;
Process proc () {
        Create (win1, attribut_list) ;
        ...
        compute (mat) ;
        Draw (win1.pdraw, mat, 150, 150, 0) ;
        ...
        Close (win1) ;
        }
```

2.2. Génération.

2.2.1. Fenêtres réelles.

La spécification de chaque fenêtre réelle donne lieu à la génération d'un programme de gestion de cette fenêtre. Ce programme deviendra à l'exécution, un processus qui réalisera les opérations effectives sur cette fenêtre. L'utilisation d'un processus facilite les communications avec d'autres fenêtres et le noyau de l'interface graphique (e.g. sous UNIX : utilisation de socket, de mémoire partagée).

2.2.2. Fenêtres virtuelles.

A chaque fenêtre réelle doit correspondre une spécification virtuelle. Pour que la représentation soit la plus fidèle possible, l'outil de conception doit générer une spécification de fenêtre virtuelle à partir de la spécification d'une fenêtre réelle. Cette spécification de fenêtre virtuelle sera inclue dans le programme parallèle sous forme de structure de données.

Dans une version minimale, une fenêtre virtuelle peut être constituée des champs suivants:

- **Type** : type de la fenêtre (global, local, individuel).
- **Propriétaire** : identificateur du processus qui a créé la fenêtre virtuelle.
- **Lien** : correspondance avec la fenêtre réelle.
- **Lien** : correspondance avec la fenêtre réelle.
- **Compteur d'états ::** compteur donnant le numéro du dernier état. Il est incrémenté chaque fois qu'un champ de la fenêtre virtuelle est mis à jour.
- **Numéro de mise à jour** : numéro du dernier état qui a été transmis à la fenêtre réelle. Il est mis à jour à chaque de transmission d'état à la fenêtre réelle.
- **Zones d'affichage** : servent à afficher du texte.

- **Zones de dessin** : matrices de pixels.
- **Zones de saisie** : zone pour recevoir des données de l'utilisateur en temps réel.

Dans un environnement couleur, chaque champ devra porter au moins un attribut supplémentaire qui est la couleur.

Exemple.

La fenêtre virtuelle de la figure 3 peut avoir la structure minimale suivante :

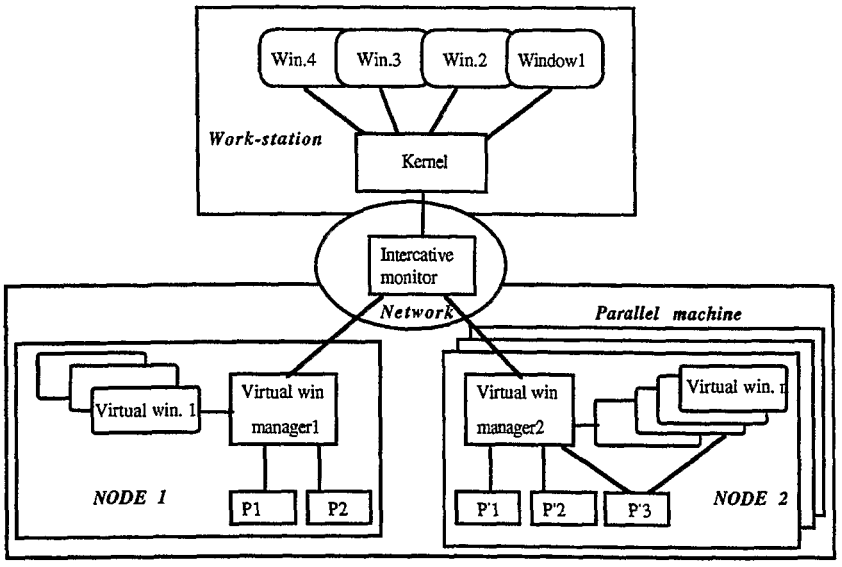
```
struct window1 {
        char            type ;
        Proc_identifier         owner ;
        int             win_link ;
        int             ctr ;
        int             last_view ;
        Pannel                  pdisplay [LINES] ;
        Bitmap                  pdraw [WIDTH][HIGHTH] ;
        } win1;
```

On notera que les boutons sont réservés au seul usage de l'utilisateur.

2.3. Fonctionnement.

A l'exécution, nous avons les composantes suivantes organisées selon la figure 4 :

- Les processus d'application qui accèdent aux fenêtres virtuelles, directement (fenêtres individuelles) ou à travers des serveurs.
- Les processus serveurs des fenêtres virtuelles, qui en assurent la mise à jour, la cohérence, et le transfert de données vers les fenêtres réelles.
- Le moniteur interactif, qui assure le transfert effectif des données entre les serveurs et le noyau.
- Le noyau qui se charge de livrer les données aux processus fenêtres, et de recevoir les requêtes de ces fenêtres destinées aux processus d'application. Il est chargé d'assurer la gestion des fenêtres réelles.
- Les processus fenêtres qui gèrent effectivement les objets fenêtres.



-Figure 4- Composantes actives à l'exécution.

Cette section décrit le fonctionnement de chacune de ces composantes, hors mis le moniteur interactif qui est considéré comme une composante du système parallèle. Des exemples de tels moniteurs sont décrits dans [2][3][6][7]

2.3.1. Processus d'application.

Les processus d'application accèdent aux fenêtres virtuelles soit directement, soit à l'aide des opérations décrites au paragraphe 2.1.2 qui servent à envoyer des requêtes au serveur. Ces requêtes ont lieu sous forme de communication avec le processus serveur, afin faciliter la gestion des objets que sont les fenêtres virtuelles (e.g. synchronisation des mises à jour concurrentes).

Dans le cas d'une mise à jour, le champ virtuel est modifié par le serveur à partir du message du processus d'application. Si un processus attend une donnée du programmeur à travers une fenêtre virtuelle, celui-ci sera bloqué jusqu'à ce que la valeur introduite par l'utilisateur dans la fenêtre réelle parvienne au serveur. On peut pour gagner du temps inscrire les requêtes du programmeur de manière anticipée dans un fichier, où le noyau irait les récupérer.

2.3.2. Processus serveurs.

Sur chaque processeur, un processus serveur est chargé de gérer les fenêtres virtuelles. Dans le cas d'une machine à mémoire distribuée, ces serveurs doivent se concerter régulièrement pour maintenir la cohésion des fenêtres virtuelles globales. C'est à dire que lorsqu'une fenêtre globale est modifiée localement, une copie du champ concerné doit être diffusée à tous les autres serveurs. La synchronisation pourra se faire par une politique de round-robin, ou celles de la littérature [6][11].

La gestion des fenêtres virtuelles consiste pour le serveur à : répondre aux processus d'application, mettre à jour les fenêtres virtuelles, communiquer avec le noyau, et communiquer avec d'autres serveurs. La communication avec les processus d'application se fait simplement par des messages. Si le message concerne la création d'une fenêtre virtuelle, une requête est envoyée au noyau pour créer une fenêtre réelle si elle n'existe pas. Ce dernier répond en envoyant le numéro de cette fenêtre réelle.

La mise à jour d'une fenêtre virtuelle se fait de la façon suivante :

- A chaque fenêtre est associé un fichier. Chaque champ en mémoire porte la dernière valeur. A chaque mise à jour venant d'un processus, la valeur du champ est enregistrée dans le fichier, étiquetée d'un numéro chronologique. Cet enregistrement est appelé *état de la fenêtre virtuelle*.

- A chaque demande d'une donnée du programmeur, le serveur émet un message vers le noyau, étiqueté de numéro de la fenêtre réelle. En recevant la réponse, il met à jour la fenêtre virtuelle et réveille le processus qui était en attente.

La communication avec le noyau sert au serveur à transmettre à celui-ci un état de la fenêtre virtuelle, à recevoir les réponses ou requêtes du programmeur, et à transmettre les demandes de lecture formulées par les processus d'application.

Les processus serveurs ont le même rôle d'une application à l'autre. De ce fait, ils seront conçus une fois pour toute et intégrés au moniteur interactif.

2.3.3. Le noyau.

Rappelons que l'interface graphique est composée du noyau et des fenêtres. Le noyau est la composante centrale de l'interface. Il réalise les fonctionnalités suivantes :

- **Collecte et distribution des messages.** Le noyau distribue aux fenêtres les messages venant des serveurs, sachant que chaque message est étiqueté du

numéro de la fenêtre virtuelle et de la fenêtre réelle correspondante. Quand il reçoit d'une fenêtre une requête du programmeur, il la transmet dans un message au serveur désigné, ou le diffuse à tous les serveurs s'il s'agit d'une requête issue d'une fenêtre globale. La communication entre les fenêtres et le noyau doit être normalisée pour assurer la probabilité des logiciels (e.g. étiquetage des messages).

- **Report des anomalies.** Les processus d'application accédant en concurrence à l'interface graphique, des conflits peuvent survenir. Par exemple un processus peut envoyer par erreur un message à une fenêtre locale à un autre processeur, ou à une fenêtre qui n'existe pas. La gestion de telles anomalies peut devenir lourde. Une solution simple consiste à en signaler l'occurrence dans une fenêtre de contrôle dédiée, et à en garder la trace. Dans le cas où un processus attend un message d'une fenêtre qui n'existe pas, la fenêtre de contrôle peut s'y substituer, mais malheureusement cela devient vite prohibitif. Nous préconisons que dans cette situation, le noyau qui connaît les fenêtres réellement actives renvoie un message d'erreur au serveur adéquat afin de débloquer le processus en attente. La gestion de telles anomalies est nécessaire pour garantir la fiabilité de l'interface graphique virtuelle.

- **Gestion des fenêtres.** Le noyau crée les fenêtres suite aux requêtes des processus, ou du programmeur à travers la fenêtre de contrôle. Il assure aussi la destruction des fenêtres à l'initiative des processus ou de l'utilisateur. Cette stratégie facilite la gestion des anomalies du fait que le noyau sait quelles sont les fenêtres actives.

On voit bien que, tout comme les serveurs, le noyau est un processus complexe, dont il faut naturellement épargner la conception au programmeur d'application. De plus le rôle du noyau est le même d'une application à une autre. Il sera donc un processus pré défini, indépendant des données communiquées.

2.3.4. Les fenêtres.

Nous avons dit qu'un processus doit gérer chaque fenêtre de l'interface graphique. Avant de discuter de l'intérêt de cette solution, nous allons en décrire le principe.

Le rôle du processus fenêtre est d'établir la correspondance entre les champs des fenêtres virtuelles et réelles. Tout message destiné à mettre à jour un champ doit être étiqueté du numéro chronologique de ce champ et le type d'opération à effectuer, étant entendu qu'il y a une bijection entre les champs virtuels et les champs réels. Le processus fenêtre vérifie la validité de cette opération avant de

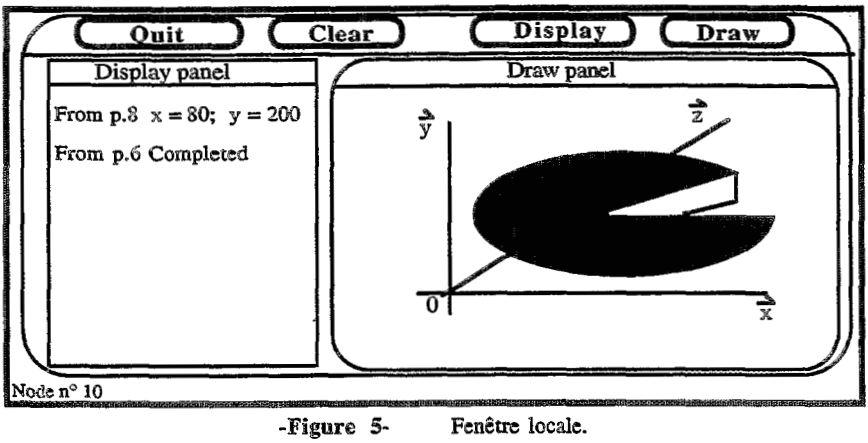
l'effectuer en appelant la fonction adéquate définie par le programmeur à la conception de l'interface graphique réelle.

La fenêtre réelle doit être rafraîchie automatiquement, périodiquement, ou encore à la demande (du programmeur ou du processus d'application) avec les états successifs de la fenêtre virtuelle qui sont stockés dans un fichier avons nous dit. Pour cela, chaque fenêtre virtuelle dispose du numéro du dernier état qui a été transmis à la fenêtre réelle. Pour le rafraîchissement automatique, la fenêtre réelle est mise à jour chaque fois qu'un champ de la fenêtre virtuelle est modifié. On peut prévoir des risques de congestion dans le circuit de communication entre la machine parallèle et la station graphique. Le rafraîchissement périodique se fait à une fréquence fixée par le programmeur.

Une fenêtre peut être sollicitée par quatre types d'interlocuteurs : les processus parallèles, l'utilisateur, le système d'exploitation et d'autres fenêtres. Cette sollicitation peut se faire directement ou à travers le noyau (notification). Elle peut être gérée efficacement à travers la technique éprouvée des événements (interruptions). D'où l'intérêt des processus pour gérer les fenêtres. En d'autres termes, l'interface graphique est elle même un système parallèle.

Exemple.

L'exemple de la figure 5 illustre une fenêtre locale du processeur N° 10, sur laquelle, les processus 8 et 6 ont affiché des messages; de même qu'une représentation en trois dimensions a été réalisée.



-Figure 5- Fenêtre locale.

III. Mise en oeuvre.

Les problèmes que l'on peut rencontrer lors de la mise en oeuvre du modèle d'interface que nous venons de décrire sont multiples, et leurs solutions relèvent souvent de la programmation. Nous allons simplement aborder quelques uns des plus importants et montrer comment les résoudre compte tenu de notre jeune expérience sur le sujet.

3.1. Communication.

Le problème de communication se pose à tous les niveaux. Entre le serveur et le processus, certains auteurs comme [6] proposent que le serveur ait un canal avec chaque processus, sachant que visiblement certains ne seront pas utilisés et encombrerait la mémoire. Nous suggérons plutôt le passage de bout de lien, qui consiste à étiqueter chaque message de l'adresse du canal sur lequel il sera reçu. Nous avons efficacement utilisé cette technique lors de la conception de CDS [8][9]. Elle rajoute quelques octets cependant au message (4 dans CDS).

La communication entre les processus fenêtres et le noyau peut se faire à travers la mémoire partagée, ou des fichiers. Sous UNIX, une panoplie de techniques existe (socket, pipe, fichier, mémoire partagé, etc.).

Enfin, la communication entre le noyau et le moniteur interactif, et entre ce dernier et les serveurs, se fera à travers un réseau.

3.2. Réduction du flow de donnée.

Le flow et la quantité de données transférées peuvent conduire à des congestions dans les circuits de communication. Une attention particulière sera donc apportée à la minimisation des échanges. Par exemple dans le cas d'un bitmap, on n'enregistrera pas la modification des états pixel par pixel, mais de préférence en utilisant des vecteurs d'états. Les messages statiques peuvent être sélectionnés, numérotés, et stockés sur le terminal graphique, de sorte qu'on n'en transmettra que le numéro. Le fait de limiter l'état d'une fenêtre virtuelle au seul champ qui a été modifié, permet de réduire considérablement l'information enregistrée. En particulier, lors du transfert vers la fenêtre réelle, seul le champ modifié sera communiqué.

3.3. Conception du moniteur interactif.

L'exemple du moniteur interactif de CDS [7][8][9] peut servir de point de repère. Il est constitué d'un processus serveur sur chaque noeud de la machine parallèle, et sur chaque machine traversée par la voie de communication. Ces processus gèrent un flot bidirectionnel. Plus précisément pour simplifier la gestion des communications, à chaque direction est associé un processus. Le bus de contrôle du SuperNode est utilisé pour véhiculer les messages. La mise en oeuvre de ce moniteur a nécessité de modifier certaines fonctions de communication du système pour pouvoir diriger les messages selon leur type. Nous avons rendu équitable la perturbation causée par le moniteur aux processus d'application en exécutant celui-ci au même niveau de priorité que ces processus.

3.4. Génération des fenêtres virtuelles.

Les outils existant actuellement ne permettent de générer qu'un prototype du programme de gestion de fenêtre. Le programmeur doit ensuite intervenir manuellement pour y ajouter les fonctions de gestion de la fenêtre. Pour générer la description de la fenêtre virtuelle à partir de ce prototype, il faut concevoir un outil qui analyserait ce prototype en extrayant les éléments décrivant la fenêtre réelle, pour déduire la structure de la fenêtre virtuelle. Cela est heureusement facilité par le fait que ces prototypes ont des formes standard facilement exploitables avec des outils comme LEX et YACC.

Conclusion.

Le modèle et la méthode de conception d'interface graphique que nous venons d'exposer permettent de créer des outils dynamiques, singularisés, contrôlées par les processus et reflétant de ce fait le parallélisme de l'application. En particulier, la gestion de la visualisation est intégrée au niveau des processus d'application.

Un des avantages certains de notre modèle est de faciliter le play-back de l'exécution. En effet, les vues des fenêtres virtuelles étant enregistrées à chaque modification, le play-back consisterait à restituer ces vues. Pour cela, il suffit d'un serveur dédié au play-back, s'exécutant sur la station graphique, qui permettrait de lire ces vues, pour les communiquer au noyau; et le fonctionnement des fenêtres se ferait comme en temps réel.

Mieux encore, en se basant sur l'idée du play-back, on peut envisager une exécution dans laquelle les fenêtres réelles ne sont pas activées. Seules les

fenêtres virtuelles sont actives. Puis, à partir des vues collectées, une animation en play-back de l'exécution sera effectuée. Différer ainsi la visualisation permet de gagner du temps, sans cependant la détacher du programme parallèle.

Le travail que nous venons de présenter est une étude dont les principaux points ont été validés par des travaux antérieurs. Il s'agit principalement du moniteur interactif, et de la communication et la synchronisation entre fenêtres réelles. Cette synchronisation consiste à contrôler l'affichage sur plusieurs fenêtres à partir d'une fenêtre quelconque. Les deux principales difficultés liées à l'implémentation d'une interface graphique virtuelle sont la nécessité d'un disque sur la machine parallèle, et l'enregistrement efficace des vues. La notion de fenêtre virtuelle est en cours d'expérimentation à l'Institut Africain d'Informatique où elle a été élaborée.

Références bibliographiques.

[1] Applebe W., McDowell C.
Integrating Tools for Debugging and Developping Multitasking Programs
pp 78-87. Proc. of the ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging, Mays
5-6 Univ. of Wisconsin, Madison. Wisconsin 53706

[2] Aspnäs M. Längbacka T.
A monitoring System for a Transputer-Based Multiprocessor
Proc. of Transputing'91, Welch P. et al., eds, IOS Press, 1991

[3] Bemmerl T., Lindhof R., Treml T.
The Distributed Monitor System of TOPSYS
Dep. of Computer Science TU Munich, Lehrstuhl für Rechnertehnik und Rechnerorganisation
Arcisstr. 21, 8000 Münichen 2, FRG

[4] Couch A.L., Krumme D.W.
An Integrated Debugging, Analysis, and Visualization Environment for Large Scale Multiprocessors
Proc. of Supercomputer Debugging Workshop'91, Albuquerque, New Mexico, USA, Nov 14-16, 1991

[5] Leblanc T.J., Mellor-Crummy J.M., Fowler R.J.
Analyzing Parallel Program Execution Using Multiple Views
Journal of Parallel and Distributed Computing 9, PP. 203-217, 1990

[6] Miller B.P., Choi J.D.
Breakpoints and Halting in Distributed Programs
Proc. Eighth Int'l Conf. Distributed Computing Systems, CS Press, Los Alamitos Calif. 1988, PP. 316-323

[7] Moukèli P.
Présentation d'un Moniteur de Communication sur le Bus de Contrôle du SuperNode,
La Lettre du Transputer, Sept. 1991, PP. 33-50

[8] Moukèli P.
Designing CDS : An On-Line Debugging System for the C_NET Programming Environment
Proc. of Supercomputer Debugging Workshop'91, Albuquerque, New Mexico, USA, Nov 14-16, 1991

[9] Moukèli P.
CDS : Un Système de Mise au Points de Programmes C_NET sur le SuperNode,
La Lettre du Transputer et des Calculateurs Distribués , Sept. 1992, PP. 43-58

[10] Moukèli P.
Un Système de Construction de Points d'arrêt Globaux pour les Programmes Parallèles,
CARI 92, Yaoundé Cameroun, 14-20 Octobre

[11] Rigsbee P.A.
X Window System Interface for CDBX
Proc. of Supercomputer Debugging Workshop'91, Albuquerque,
New Mexico, USA, Nov 14-16, 1991
[12] Young B.
bdb : A Library Approch to Writing a New Debugger
Proc. of Supercomputer Debugging Workshop'91, Albuquerque,
New Mexico, USA, Nov 14-16, 1991