(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Numéro de publication : **0 470 881 A1**

(12) DEMANDE DE BREVET EUROPEEN

(21) Numéro de dépôt : **91402070.6**

(51) Int. Cl.[5] : **G09G 1/00,** G06F 3/14

(22) Date de dépôt : **24.07.91**

(30) Priorité : **02.08.90 FR 9009884**

(43) Date de publication de la demande :
12.02.92 Bulletin 92/07

(84) Etats contractants désignés :
AT BE CH DE DK ES FR GB GR IT LI LU NL SE

(71) Demandeur : **BULL S.A.**
121, Avenue de Malakoff
F-75116 Paris (FR)

(72) Inventeur : **Bouchet, Alain**
6, Rue croix Saint-Benoit
F-95130 Franconville (FR)
Inventeur : **Marie-Sainte, Alain**
6 esplanade des Abymes
F-94000 Creteil (FR)

(74) Mandataire : **Debay, Yves et al**
BULL S.A., 121, Avenue de Malakoff,
PC-8M006
F-75116 Paris Cédex (FR)

(54) Procédé de retaillage ou de déplacement de fenêtres.

(57) Procédé de retaillage ou de déplacement de fenêtres dans une application "Windows", caractérisé en ce qu'il consiste à interposer entre "Windows" (1) et les différentes applications (20, 21, 22) tournant sous "Windows" des filtres (4) interceptant des messages particuliers, à traiter par une application spécifique (5) ces messages et à renvoyer à "Windows" un message neutre "WM-ENTERIDLE" qui ne provoque aucune action de la part de "Windows" et ne bloque pas le fonctionnement des autres applications.

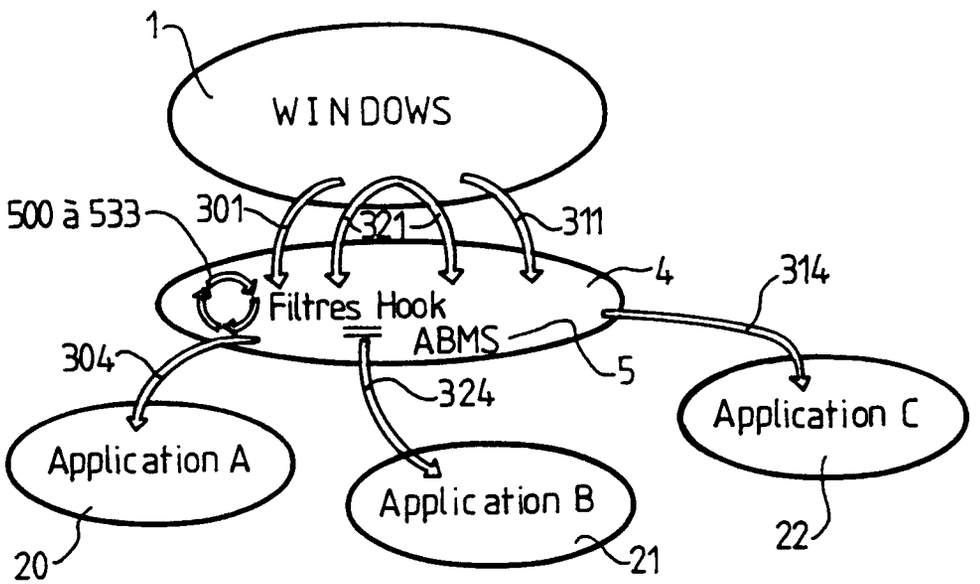

FIG.1

EP 0 470 881 A1

L'invention concerne un procédé de retaillage ou de déplacement de fenêtres.

Un procédé de retaillage ou de déplacement de fenêtre est connu, notamment par le logiciel multitâches multifenêtres "Windows" de Microsoft. Ce type de logiciel présente l'inconvénient dans le cas du retaillage ou du déplacement d'une fenêtre de bloquer l'évolution des applications qui tournent dans les autres fenêtres.

La présente invention a pour but de pallier cet inconvénient.

Ce but est atteint par le fait que le procédé de retaillage ou de déplacement de fenêtres dans une application "Windows" consiste à interposer entre "Windows" et les différentes applications tournant sous "Windows" des filtres interceptant des messages particuliers, à traiter par une application spécifique ces messages et à renvoyer à "Windows" un message neutre qui ne provoque aucune action de la part de "Windows" et ne bloque pas le fonctionnement des autres applications.

Selon une autre particularité, le traitement effectué par l'application spécifique consiste à traiter chacun des évènements particuliers pouvant intervenir et correspondant potentiellement à un évènement de déplacement de la fenêtre ou de retaillage d'une fenêtre, et au cours de ce traitement à faire appel à des fonctions spécifiques permettant d'une part d'initialiser un certain nombre de paramètres W_Top, W_Right, W_Bottom, W_Left, W_Caption, Frm_CurPos, h_WndCurr conservés jusqu'à l'occurence d'un évènement ultérieur qui provoquera la fin du traitement.

Selon une autre particularité, les paramètres conservés sont constitués par :

- "h_WndCurr" = "h_WndCurrent, qui est une variable permettant de savoir qu'une action a été engagée sur une fenêtre quelconque ;
- "b_LoadedIcon, pour mémoriser qu'une icône a été chargée ;
- "h_WndMenu", qui est la variable permettant de savoir, dans le cas d'une icône, si on est dans une phase de déroulement de menu ou de déplacement d'icône ;
- "h-OldCursor", qui est la variable permettant de mémoriser un identifieur du curseur actif de Windows avant le lancement de l'application et le remplacement par le curseur spécifique de l'application
- "w_CXScreen", "w_CYScreen", "w_CXframe", "w_CYMinHeigth", "w_CXMinWidth", qui sont respectivement les variables de coordonnées de la fenêtre en cours de retaillage, de cadre et de largeur mimimum ;
- "Frm_CurPos", qui est la variable de position courante du cadre ;
- "Mse_CurPos", qui est la variable de position courante de la souris ;
- "w_Left", "w_Top", "w_Right", "w_Bottom", "w_Caption", qui sont les variables de direction utilisées dans le calcul des coordonnées de la nouvelle fenêtre et les tests de direction ;
- "Wnd_StartPos", qui est la variable de définition de la position initiale de la fenêtre ;
- "Mse_StartPos", qui est la variable de définition de la position initiale du curseur ;
- "b_Cursor" et "b_LoadedIcon", qui sont des variables booléennes supposées fausses au départ.

Selon une autre particularité, les filtres sont en nombre de 2. Un premier "WM_GETMESSAGE" permettant de recevoir les messages postés par les interruptions hardware, tels que :

- WM_NCLBUTTONDOWN
- WM_MOUSEMOVE
- WM_KEYDOWN
- WM_SYSKEYDOWN

et un deuxième filtre "WH_CALLWNDPROC" permettant de filtrer et de recevoir les messages envoyés à la méthode, tels que :

- WM_SYSCOMMAND
- WM_ACTIVATEAPP
- WM_NCACTIVATE
- WM_ACTIVATE.

Selon une autre particularité, l'interception d'un message déclenche le traitement d'un programme spécifique à chaque message et fait intervenir les paramètres conservés et des fonctions principales de traitement, telles que :

- ABMSInit pour initialiser un traitement,
- ABMSMove pour effectuer le déplacement de la fenêtre, et
- ABMSEnd pour terminer un traitement, qui elles-mêmes font appel à des fonctions utilitaires.

Selon une autre particularité, la fonction ABMSInit est mise en oeuvre uniquement pour le traitement des messages "WM_NCLBUTTONDOWN" et "SC_SIZE" qui est un message dépendant de "WM_SYSCOMMAND".

Selon une autre particularité, la fonction ABMSEnd est mise en oeuvre uniquement pour le traitement des messages "WM_LBUTTONUP" et "VK_ESCAPE", VK-RETURN" qui dépendent de "WM_SYSKEYDOWN".

Selon une autre particularité, la fonction ABMSMove est mise en oeuvre uniquement pour le traitement des messages "WM_MOUSEMOVE, MK_LEFT, VK_UP, VK_RIGHT et VK_DOWN" qui sont des messages dépendant de "WM_SYSKEYDOWN".

Selon une autre particularité, la fonction ABMSInit consiste à approprier les messages souris par l'instruction "SETCAPTURE", puis à regarder si la fenêtre est une fenêtre fille d'une fenêtre mère, c'est-à-dire inscrite dans la fenêtre mère pour limiter les mouvements de la fenêtre fille, puis à initialiser les

coordonnées initiales de la fenêtre et la position du curseur pour devenir les coordonnées courantes et à initialiser les paramètres caractéristiques de la fenêtre (w_CXframe, w_CXMinWidth, w_CYMinHeigth).

Selon une autre particularité, le procédé comprend les étapes suivantes :

– passage en mode actif des filtres ;

– mémorisation de l'identifieur de la fenêtre "h_Wnd", destinataire du message ;

– mémorisation du type d'action résultant de la zone de clic, cette mémorisation se faisant par la fonction HitTest et la fonction ABMSInit ;

– appropriation des messages souris ultérieurs pour le compte de la fenêtre en cause ;

– premier dessin du cadre fantôme autour de la fenêtre par la procédure InvertBlock ;

– neutralisation du message par remplacement et substitution d'un message "WM_ENTERIDLE" qui n'est pas traité et ne provoque aucune action de la part de Windows.

Selon une autre particularité, le procédé comprend les étapes suivantes :

– calcul des coordonnées finales de la fenêtre par la procédure ABMSComputNewPos, effacement du fantôme par InvertBlock ;

– dessin de la fenêtre en position finale par la fonction ABMSMove ;

– remise à O des paramètres de mémorisation par la fonction ABMSEnd.

Selon une dernière particularité, le procédé comprend en outre les étapes suivantes :

– abandon de la propriété des messages souris par l'instruction "Release Capture" ;

– passage du filtre en mode passant.

D'autres caractéristiques et avantages de la présente invention apparaîtront plus clairement à la lecture de la description ci-après faite en référence aux dessins annexés dans lesquels :

– la figure 1 représente le chemin de principe général de l'invention ;

– la figure 2 représente le principe de fonctionnement des filtres de l'invention ;

– la figure 3 représente un tableau des scénarii possibles ;

– la figure 4 représente l'organigramme de fonctionnement du programme d'interprétation suite à la mise en place de la fonction de filtrage ;

– les figures 5 à 12 représentent le détail des différentes étapes représentées dans l'organigramme de la figure 4 ;

– la figure 13 représente l'art antérieur à l'invention.

L'invention concerne un perfectionnement à un logiciel multitâches et multifenêtres fonctionnant sur un matériel informatique comportant une unité centrale de traitement ou processeur, un moniteur d'affichage, une ressource clavier et une souris.

Dans l'art antérieur de la figure 13, représentant le fonctionnement d'un logiciel permettant le déroulement de plusieurs applications avec, pour chaque application, une ou plusieurs fenêtres, on peut constater un certain nombre d'inconvenients. Ainsi, dans un environnement "Windows" ou OS2, le noyau (1) du logiciel "Windows" émet des messages (30, 31) en direction des différentes applications ou tâches (T1, T2, T3), respectivement (20, 21, 23). Le noyau (1) de "Windows" permet d'adresser des messages (30, 31) aux tâches activées, par exemple dans le cas de la figure 13 les tâches (T1) et (T3), et d'empiler ces messages adressés à la tâche respective dans une file d'attente respective (Q1, Q3). Le noyau (1) de "Windows" attribue le processeur de traitement à l'application qui va venir gérer les messages de la file et les traiter, et lorsque la file est épuisée, la tâche rend le processeur au noyau (1) de "Windows" pour permettre l'attribution du processeur à une autre application activée. Chaque application (T1, T3) comporte une fenêtre associée (F1, F3). Chaque fenêtre constitue un objet visible ou non. A chaque fenêtre est associée une méthode qui constitue le mode d'emploi ou de réaction de la fenêtre aux divers messages qui concernent la fenêtre de l'application. Cette méthode est un programme qui traite chacun des messages et réalise des actions en conséquences. Ce programme peut choisir de traiter certains de ces messages ou de laisser traiter certains autres par "Windows". Dans ce dernier cas, le programme de l'application prevoit l'envoi d'un message (32) vers une fonction (13) du noyau (1) de "Windows", fonction qui est appelée "default Windows Proc". Le noyau (1) de "Windows" sait reconnaitre ces messages particuliers et effectue, par exemple dans le cas du retaillage d'une fenêtre ou du déplacement de celle-ci un traitement standard. Ainsi, un opérateur que veut retailler une fenêtre, va agir sur la "souris" du système qui génère par les interruptions du Hardware un certain nombre de messages. Ces messages vont être envoyés et empilés dans la file, par exemple (Q1), de l'application concernée. Le programme, lors du traitement de la file, passe ces messages à la méthode de la fenêtre qui cherche à savoir où se trouve la souris, c'est-à-dire si elle se trouve sur un bord, au milieu, ou sur une zone menu de la fenêtre. En fonction de cet endroit, la méthode génére un certain nombre de messages distincts.

Si la souris est sur un bord de la fenêtre, l'interruption qui en résulte génère le message "WM_LBUTTONDOWN". Lorsque la méthode récupère ce message, elle l'envoie à la fonction (13) "Def Window Proc" et rend la main à "Windows". "Windows", détectant ce message particulier, s'enquiert de savoir sur quelle zone de la fenêtre il a été obtenu, et selon le cas, génère un certain nombre de messages, tels que "WM_SIZE" (correspondant à un retaillage de la fenêtre) ou "WM_MOVE" (correspondant à un déplacement de la fenêtre) et se met en attente

des messages suivants provenant de la souris de façon à pouvoir les traiter immédiatement.

Les messages (30, 31) envoyés peuvent être soit des messages système envoyés par le noyau (1) de "Windows" directement à une application, soit des messages provenant du Hardware, souvent après une interruption, par exemple messages souris ou messages clavier. Lorsque la méthode a envoyé un message à la fonction "Def Wnd Proc", et tant que les autres informations concernant le retaillage ou le déplacement de la fenêtre permettant à la fonction (13) de terminer le traitement du message ne sont pas parvenues à cette fonction, le processeur continue d'exécuter cette fonction, n'en sort plus et il se met à attendre les autres messages. Ceci a pour inconvénient de bloquer le processus de traitement des autres tâches. Cet inconvénient est particulièrement gênant pour des applications temps réel, par exemple, de surveillance en temps réel ou de pilotage d'alarme. Dans ce type d'application, il peut arriver qu'une manoeuvre de retaillage ou de déplacement de fenêtre par clavier soit lancée et que l'opérateur dérangé bloque toutes les autres applications pendant tout le temps où il n'a pas eu la possibilité de terminer son action de taillage ou de déplacement d'une fenêtre, en intervenant sur le clavier ou sur la souris.

L'invention vise donc à éliminer cet inconvénient et propose l'architecture de la figure 1 dans laquelle, entre les différentes applications (20, 21, 22) et le noyau (1) de "Windows", est placé un filtre (4), accroché au noyau "Windows", lequel lors du passage de certains messages va assurer le traitement de ces messages par une application ABMS (5). Ainsi, les messages (301, 321, 331) envoyés par le noyau "Windows" vers les applications A, B, C (20, 21, 22) passent dans le filtre (4), et selon l'activation ou la non-activation sont transmis ou traités par ABMS. Dans le cas de l'activation du filtre, certains types de messages sont transmis sous forme de messages (304) à l'application A, (324) à l'application B et (314) à l'application C. Dans le cas où l'un des messages (301, 321, 313) correspond à un évènement spécifique (50, 51, 52, 53), comme représenté aux figures 2 et 4, ce message est intercepté et traité par l'application ABMS (5), comme on va le voir par la suite.

L'invention repose donc sur l'utilisation des filtres de message fournis par Windows, ces filtres permettant à une application quelconque d'être informée du passage de tous les messages correspondant à un type d'événement particulier et de les modifier ou les substituer avant qu'ils ne parviennent à leur véritable destinataire. L'application ABMS (5) utilise donc deux de ces filtres spécifiques. Deux filtres sont nécessaires pour couvrir tous les messages possibles dans les cas qui nous intéressent, parce que Windows classe tous les messages par familles de messages et les messages qui nous intéressent appartiennent à deux de ces familles. Un filtre "WM_GETMESSAGE" (400, fig. 5) permet de recevoir les messages postés par les interruptions hardware, c'est-à-dire la souris ou le clavier, messages qui auraient été normalement déposés dans la queue de l'application. Un autre filtre "WH_CALLWNDPROC" (401, fig. 5) permet de filtrer et de recevoir tous les messages qui seraient envoyés directement à la méthode de l'application. Ces filtres sont installés lors de l'initialisation de l'application ABMS, cette installation étant représentée par la référence (40) aux figures 2 et 5, et s'effectuant par la fonction "SETWINDOWSHOOK". La cloture de l'application ABMS retire les filtres, ce retrait étant représenté par la référence (41) aux figures 2 et 5 et effectué par la fonction "UNHOOKWINDOWSHOOK". Ces filtres installés peuvent être passants, c'est-à-dire n'affectant pas les messages ou actifs, c'est-à-dire provoquant un traitement particulier survenant d'un certain évènement spécifique. Dans le cas où le filtre est passant, les messages (301, 321, 311) arrivant dans le filtre seront transmis aux applications sous forme de message distribué (304, 324, 314), et dans le cas où le filtre est actif, il enverra le message à l'application ABMS pour traitement.

Les options prises en compte par l'application ABMS peuvent être décomposées en divers scénarii qui sont classés dans un tableau à trois dimensions (cf figure 3) représentant les actions, les moyens utilisés et les états de la fenêtre manipulée. La réalisation de l'application ABMS a nécessité la solution d'un certain nombre de difficultés, certaines générales et rappelées par la suite, d'autres spécifiques d'un scénario et indiquées dans la case correspondante de la figure 3.

Les difficultés générales concernent la terminaison de l'action lors de l'annulation par la touche ESC. A ce moment, la restauration de l'état initial suppose la mémorisation d'un certain nombre d'informations constituées par des variables rémanentes dans la zone de données de l'application ABMS. Une autre difficulté est l'interpénétration des scénarii clavier/souris. En effet, la frontière entre le scénario clavier et souris n'est pas étanche, une action de reprise avec l'un de ces moyens peut être poursuivie avec l'autre à tout instant. ABMS prend en compte cette souplesse. Chaque message traité fait évoluer le contexte général de l'application en modifiant le contenu des variables ou en jouant sur des paramètres définis dans la zone de données d'ABMS, et laisse les variables dans un état clairement défini susceptible d'être repris pas n'importe quel autre message. Ces variables sont représentées par :

- "h_WndCurr", qui est une variable permettant de savoir qu'une action a été entamée sur une fenêtre ;
- "h_WndMenu", qui est la variable permettant de savoir, dans le cas d'une icône, si on est dans une phase de déroulement de menu ou de déplace-

ment d'icône ;

– "h_OldCursor", qui est la variable permettant de mémoriser un identifieur au curseur actif de Windows avant le lancement de l'application et le remplacement par le curseur spécifique de l'application

– "w_CXScreen", "w_CYScreen", "w_CXframe", "w_CYMinHeigth", "w_CXMindWidth", qui sont respectivement les variables de coordonnées de la fenêtre en cours de retaillage, de cadre et de largeur mimimum ;

– "Frm_CurPos", qui est la variable de position courante du cadre ;

– "Mse_CurPos", qui est la variable de position courante de la souris ;

– "w_Left", "w_Top", "w_Right", "w_Bottom", "w_Caption", qui sont les variables de direction utilisées dans le calcul des coordonnées de la nouvelle fenêtre et les tests de direction ;

– "Wnd StartPos", qui est la variable de définition de la position initiale de la fenêtre ;

– "Mse_StartPos", qui est la variable de définition de la position initiale du curseur ;

– "b_Cursor" et "b_LoadedIcon", qui sont des variables booléennes supposées fausses au départ.

Dans le cas du retaillage d'une fenêtre qui n'est pas une icône effectué uniquement par la souris, l'opérateur doit effectuer les opérations suivantes :

– il positionne le curseur de la souris sur un des bords de la fenêtre, appuie sur le bouton gauche et le maintien enfoncé. Il déplace ensuite la souris, ce qui a pour effet de déglacer sur l'écran un des bords du cadre représentant le fantôme de la fenêtre. Lorsque l'utilisateur estime avoir atteint la taille désirée, il relâche le bouton gauche et la fenêtre se redessine dans le cadre qui vient d'être redéfini. Au début de l'opération, le filtre installé par ABMS est initiallement dans l'état passant. Tous les messages de déplacement de la souris émis par le noyau (1) de "Windows" sont distribués. Lorsque l'opérateur appuie sur le bouton gauche de la souris, le curseur se trouvant sur le bord de la fenêtre, "Windows" génère un message "WM_NCLBUTTONDOWN" dont les paramètres contiennent l'identifieur de la fenêtre et une valeur définissant la zone sur laquelle l'événement s'est produit. Les actions de ABMS sont alors les suivantes :

  – passage en mode actif, mémorisation de l'identifieur de la fenêtre HWND, destinataire du message et coordonnées initiales ;

  – mémorisation du type d'action résultant de la zone de clic (retaillages gauche, droit, haut, bas, diagonale) ; cette mémorisation se fait par la portion de programme ABMSHit TEST, portant la référence (501) sur la figure 11, et par le programme ABMSInit, portant les références (502, 503) sur la figure 7 ;

  – appropriation des messages souris ultérieurs (par l'instruction SETCAPTURE) pour le compte de la fenêtre en cause ;

  – premier dessin du cadre fantôme autour de la fenêtre par la procédure ABMSInvertBlock, représentée à la figure 8 par la référence (521), qui à chaque mouvement annule le cadre fantôme précédent et redessine le suivant au nouvel emplacement ;

  – neutralisation du message par remplacement et substitution d'un message "WM_ENTERIDLE" qui n'est pas traité et ne provoque aucune action de la part de Windows.

Lorsque le programme ABMS est lancé, une première fonction, représentée à la figure 5 par la référence (40), installe ou désinstalle la fonction filtre. L'installation de la fonction filtre se fait par la fonction " SETWINDOWSHOOK", et la désinstallation se fait par la fonction "UNHOOKWINDOWSHOOK", comme représenté sur la figure 2 par les références (40, 41). Une fois cette fonction filtre activée, tous les messages tels que "WM_SYSCOMMAND, WM_LBUTTONUP, WM_SYSKEYDOWN, WM_NCL BUTTONDOWN, WM_MOUSEMOVE" sont détournés, traités par le programme ABMSHook et neutralisés par envoi du message "WM_ENTERIDLE" de l'application à "Windows". Tous ces messages sont au départ émis par "Windows" à la suite d'une interruption clavier ou souris.

Ainsi, comme représenté à la figure 4, lorsque le message "WM_NCLBUTTONDOWN", référencé (50), est émis par "Windows", celui-ci est intercepté par le programme ABMSHook qui mémorise dans un premier temps (504) l'identifieur de la fenêtre, destinataire du message dans la variable "h_WndCurr", effectue un test pour savoir si la fenêtre est une icône, ce test étant indiqué dans le programme (50) de la figure 5 par la ligne (500), mémorise le type d'action résultant de la zone de cliquage par la portion de programme ABMSHit test (501), figure 5, représenté en détail à la figure 11 par la référence (501), et lance le sous-programme ABMSInit (502, 503), figure 5, représenté en détail par les références (502, 503) à la figure 8. Ce programme approprie les messages souris ultérieurs par le message "SETCAPTURE" pour le compte de la fenêtre en cause et détermine les coordonnées initiales de la fenêtre et de la position du curseur. La séquence (50) de traitement de "WM_NCLBUTTONDOWN" renvoie à la fin du traitement le message "WM_ENTERIDLE".

Le message suivant, envoyé par Windows, peut être un mouvement de la souris, signalé par le message "WM_MOUSE MOVE" et traité par la séquence d'instruction (51) de la figure 5 à la fin de laquelle le programme rend la main à Windows par l'envoi du message "WM_ENTERIDLE". L'autre possibilité

d'instruction est une instruction de relâchement du bouton de la souris, cette instruction étant signalée par le message Windows "WM_LBUTTONUP", dont la séquence de traitement (52) est représentée à la figure 5. Cette séquence lance le sous-programme ABMSEnd qui à la fin rend la main au noyau (1) de Windows pour l'envoi du message "WM_ENTERIDLE".

Dans le cas d'un message "WM_MOUSEMOVE" envoyé par Windows, le programme regarde si l'objet n'est pas iconique par la référence (510), figure 5 ; dans le cas où il n'est pas inconique, l'instruction (515) regarde si les variables de direction ont été initialisées et dans ce cas lance la fonction ABMSMove (511), figure 9, sinon la fonction ABMSDirection (512), figure 10, est lancée pour initialiser les variables de direction avec comme paramètres le résultat de la fonction ABMSTestDirect (513), figure 12. Dans le cas où l'objet est iconique, la séquence lance par l'instruction référencée (5111) le chargement de l'icône à la position indiquée par le curseur et la fonction ABMSLoadIcon (5100). Ensuite, la séquence rend la main à Windows en envoyant le message "WM_ENTERIDLE".

Lorsque ABMS intercepte un message "WM_SYSKEYDOWN", représenté par la référence (53) à la figure 4, le traitement d'un tel message met en oeuvre le sous-programme (53) qui commence par traiter le cas de l'actionnement d'une touche de retour chariot lors de la réception d'un message "VK_RETURN" ou d'échappement représenté par un message "VK_ESCAPE", et ensuite traite le cas des touches de déplacement à gauche "VK_LEFT", vers le haut "VK_UP", à droite "VK_RIGHT", vers le bas "VK_DOWN". Selon le type de touche enfoncée, il effectue le traitement correspondant à un des sous-programmes (530) pour le déplacement à gauche, (531) pour le déplacement vers le haut, (532) pour le déplacement à droite, (533) pour le déplacement vers le bas.

Comme on l'a expliqué ci-dessus, dès qu'un message est reçu par le filtre, celui-ci est traité, et le programme ABMS, après le traitement correspondant, rend la main au noyau (1) de Windows qui peut ainsi redonner la main à une autre application en attendant que le filtre intercepte un nouveau message dont le traitement nécessite l'intervention du programme ABMS. Le filtre reste actif jusqu'à ce que l'opérateur relâche le bouton gauche de la souris ou tape "Enter" ou "Esc". Cette action génère un message "WM_LBUTTONUP, WM_SYSKEYDOWN, VK_ENTER, VK_ESCAPE" qui prévoit le changement d'état du filtre ABMS de l'état actif à l'état passant. Les actions effectuées au cours de ce changement sont les suivantes :

- calcul des coordonnées finales de la fenêtre par la procédure ABMSComputNewPos, représentée par le sous-programme (5210) de la figure 10 ;
- effacement du fantôme par InvertBlock ;
- dessin de la fenêtre en position finale ;
- remise à zéro des paramètres mémorisés ;
- abandon de la propriété des messages souris par l'instruction "Release Capture" ;
- passage du filtre en mode passant.

Comme on vient de l'expliquer, l'application ABMS, grâce au filtre installé, reçoit les différents messages "WM_SYSCOMMAND, WM_NCLBUTTONDOWN, WM_MOUSEMOVE, WM_LBUTTONUP, WM_SYSKEYDOWN, WM_KEYDOWN, WM_ACTIVATEAPP, WM_NCACTIVATE, WM_ACTIVATE". Cette application substitue à ces messages, à l'intention de Windows, un message neutre tel que "WM_ENTERIDLE" et ensuite procède au traitement de ces messages en faisant appel à différentes fonctions principales de traitement appelées lorsque l'on détecte qu'un traitement est initialisé. Ces fonctions principales de traitement sont :

- la fonction ABMSInit pour initialiser un traitement lors de la réception, par exemple d'un premier "SC_MOVE" ou d'un "SC_SIZE" dans un "WM_SYSCOMMAND" ;

Cette fonction ABMSInit consiste à approprier les messages souris par l'instruction "SETCAPTURE", puis à regarder si la fenêtre est une fenêtre fille d'une fenêtre mère, c'est-à-dire inscrite dans la fenêtre mère pour limiter les mouvements de la fenêtre fille, puis à initialiser les coordonnées initiales de la fenêtre et la position du curseur pour devenir les coordonnées courantes et à initialiser les paramètres caractéristiques de la fenêtre (w_CXFrame, w_CXMinWidth, w_CYMinHeight).

- puis la fonction ABMSEnd et la fonction ABMSMove.

En plus de ces trois fonctions principales, il y a des fonctions utilitaires appelées en différents endroits des fonctions principales et qui servent à effectuer les traitements utilitaires. Ces fonctions sont :

- la fonction ABMSComputNewPos qui sert, lorsqu'on lui donne les coordonnées de la souris, à calculer et mettre à jour les coordonnées de la fenêtre en tenant compte des variables globales et rémanentes. Cette fonction ABMSComputNewPos (5210), figure 9, permet de calculer les nouvelles positions à partir des paramètres W_left, W_Caption, W_Top, W_Right, W_Bottom et les coordonnées de la souris et du cadre de la fenêtre, ainsi qu'à partir des coordonnées minimales en hauteur et en largeur des fenêtres. Elle teste aussi les limites de ces calculs pour s'assurer qu'une fenêtre fille ne sort pas d'une fenêtre mère.
- la fonction ABMSDirection (512), figure 10, sert à déterminer la forme du curseur et à initialiser les variables de direction en fonction des messages

reçus. Cette fonction ABMSDirection (512), figure 10, par exemple dans le cas où le paramètre renvoyé par ABMSTestDirection est D_Top, positionne le paramètre W_Top à 1 et accroche la position du curseur au bord supérieur de la fenêtre par l'instruction référencée (5120), ensuite la séquence effectue un test sur le paramètre W_Left pour savoir si W_Left est initialisé et dans le cas affirmatif, envoie le message "D_TTOBDBLEARROW" pour demander l'affichage du curseur incliné à 45° vers le haut et la droite. Si W_Left n'est pas initialisé, la séquence regarde si W_Right est initialisé et dans ce cas envoie le message "D_BTOTDBLEARROW" pour demander l'affichage du curseur incliné vers le haut et la gauche, et dans le cas où le test sur W_Right est négatif, un curseur vertical est affiché par l'exécution du message "D-VERTDBLEARROW".

– la fonction ABMSHitTest (501), figure 11, sert à déterminer à l'initialisation dans quelle zone on a cliqué pour initialiser, dans le cas des mouvements souris, les variables de direction. Cette fonction est appelée par ABMSInit dans le cas où on reçoit un "WM_NCLBUTTONDOWN" et permet d'initialiser les variables W_Top, W_Right, W_Bottom, W_Left et W_Caption.

Ainsi, comme on peut le voir à la figure 11 lorsque l'on reçoit un message HT_TopRight, qui indique que le coin haut droit a été cliqué par la souris, la fonction ABMSHitTest met à 1 les variables W_Top et W_Right qui permettront par la suite, lors de la réception d'un nouveau message de savoir qu'il y a eu une initialisation dans le coin haut droit de la fenêtre, et que par conséquent, on doit traiter le déplacement de la fenêtre ou le retaillage de celle-ci.

– la fonction ABMSInvertBlock, représentée par la référence (5030), figure 11, effectue le dessin du rectangle en dessinant quatre rectangles adjacents commandés par la fonction PatBlt, en fonction des paramètres hDC et des dimensions de la fenêtre représentée par exemple par P_RecFrm-.left. Cette fonction PatBlt effectue en même temps par la commande DSTINVERT l'inversion du fantôme de la fenêtre.

– la fonction ABMSLoadCursor (5031), figure 11, qui charge le curseur en fonction des paramètres envoyés qui dépendent de la position sur le bord de la fenêtre ; ainsi, si les paramètres W_Left et W_Top sont initialisés à 1, on a affaire au bord gauche haut de la fenêtre, et le curseur initialisé aura une allure de flèche orientée vers le bord droit inférieur de la fenêtre, c'est-à-dire à 45°.

– la fonction ABMSLoadIcon (5100), figure 12, qui permet lorsque l'on déplace une icône statique de remplacer le curseur par l'image de l'icône statique générée par Windows, et dans le cas où on déplace une icône dynamique de remplacer cette image d'icône dynamique par une image constituée par le curseur.

– la fonction ABMSTestDirect, représentée à la référence (513) de la figure 12, permet de déterminer le paramètre à renvoyer, par exemple D_Left (5130), en effectuant un test sur l'information reçue par l'application et envoyée par Windows, constituée par la coordonnée en X de la position de la souris représentée par P_MSEPOS.X et de comparer cette coordonnée à la valeur de la position gauche de la fenêtre, et dans le cas où la valeur de la coordonnée en X de la souris est inférieure à l'abscisse du bord gauche de la fenêtre, cette fonction retourne le message D_Left à l'application. Cette fonction est utilisée en conjonction avec la fonction ABMSDirection.

– la fonction ABMSMove (511), figure 9, permet d'effectuer le mouvement de la fenêtre en inversant le fantôme de la fenêtre par la fonction ABMSInvertBlock, représentée par la référence (5030) aux figures 9 et 11, puis ensuite de calculer la nouvelle position de la fenêtre par la fonction ABMSComputNewPos (5210), et enfin d'inverser le fantôme de la fenêtre pour la nouvelle position calculée, représentée par !a référence (5112).

– la fonction ABMSEnd (521), figure 8, termine le processus en regardant par l'instruction (5211) si la fenêtre est iconique. Si elle n'est pas iconique, on termine par une fin normale (5212), c'est-à-dire en lançant la fonction ABMSComputNewPos (5210), sinon le curseur reprend par l'instruction (5213) la position initiale qu'il avait au départ. Cette fonction permet également par la séquence (5214) de déterminer si la fenêtre en traitement est une fenêtre fille et dans ce cas de convertir les coordonnées avant un mouvement. De même cette fonction, si nécessaire, restaure par la séquence (5215) le curseur et remet par la séquence (5216) les paramètres à 0.

L'application par le filtre CalMsgHook permet de filtrer et recevoir les messages qui auraient été envoyés directement à la méthode de l'application, entre autres les messages "WM_SYSCOMMAND, WM_ACTIVATEAPP, WM_NCACTIVATE, WM_ACTIVATE". Le traitement du message "WM_SYSCOMMAND", représenté en (54), peut se décomposer en un traitement, soit d'un message "SC_MOVE", soit d'un message " SC_SIZE". Le message "SC_SIZE" positionne les paramètres W_Caption à 1, et le message de retaillage de fenêtre "SC_SIZE" provoque le traitement représenté par la référence (540). Lorsque l'application reçoit un message de retaillage de la fenêtre, elle demande (5401) la position du curseur en imposant que celle-ci soit celle de la souris. Ensuite, cette procédure lance en (5402) la fonction ABMSInit et la position du curseur (5403) est mise en accord avec l'action effectuée et en fin de traitement l'application renvoie à Windows le message "WM_ENTERI-

DLE", référencé 541, qui se substitue ainsi au message "WM_SYSCOMMAND"

De même, le filtre GetMsgHook filtre les messages "WM_NCLBUTTONDOWN, WM_MOUSEMOVE, WM_LBUTTONUP, WM_KEYDOWN, WM_SYSKEYDOWN". Lorsqu'un tel message "WM_NCLBUTTONDOWN" arrive à l'application ABMS, on regarde par l'instruction référencée 504 s'il s'agit d'un déroulement de menu ou non. S'il ne s'agit pas d'un déroulement de menu, on regarde si on travaille pour une icône par la référence 500. Si on ne travaille pas pour une icône, on regarde si les paramètres ont été initialisés par la fonction ABMSHitTest (501). Si les paramètres n'ont pas été initialisés, on lance la fonction ABMSInit (502, 503) en rangeant dans le point les coordonnées en X et en Y fournies par le paramètre message, et ensuite on envoie à Windows le message "WM_ENTERIDLE" (505) pour lui rendre la main. Dans le cas d'un message "WM_MOUSEMOVE" envoyé par windows, le programme regarde si l'objet n'est pas iconique par la référence (510), figure 5, dans le cas où il n'est pas iconique, l'instruction (515) regarde si les variables de direction ont été initialisées et dans ce cas lance la fonction ABMSMove (511), figure 9, sinon la fonction ABMSDirection (512), figure 10, est lancée pour initialiser les variables de direction avec comme paramètres le résultat de la fonction ABMSTestDirect (513), figure 12. Dans le cas où l'objet est iconique, la séquence lance par l'instruction référencée (5111) le chargement de l'icône à la position indiquée par le curseur et la fonction ABMSLoadIcor. (5100). Ensuite, la séquence rend la main à Windows en envoyant le message "WM_ENTERIDLE".

Dans le cas où l'on reçoit le message "WM_LBUTTONUP" (52) de l'application, on regarde si l'opérateur a effectué un déroulement de menu par la référence (520), sinon l'application regarde si la coordonnée de la fenêtre correspond à celle de la souris et lance la fonction ABMSEnd (521) pour ensuite envoyer à Windows le message "WM_ENTERIDLE".

Dans le cas des messages envoyés à la suite de l'actionnement d'une touche, l'application commence à la référence 534 par traiter le cas des touches d'avortement constituées par les touches d'échappement (VK_ESCAPE) et de retour chariot (VK-RETURN). Si on n'est pas dans ce cas là et que l'application a reçu par exemple le message VK_Left, l'application regarde tout d'abord par la séquence référencée 530 si l'on est iconique, et charge l'icône par la fonction ABMSLoadIcon (5100), regarde ensuite par l'instruction (5301) si les paramètres W_Caption, W_Left, W_Right ont été initialisés et dans ce cas, déplace l'icône ou donne la grandeur réelle à celle-ci par la séquence (5302). Dans le cas où on n'est pas iconique à ce moment-là, on lance la fonction ABMSMove et on donne à la position du curseur la position de la souris. Autrement, ABMS lance la fonction ABMSDirection et ensuite renvoie à Windows le message "WM_ENTERIDLE".

Le fonctionnement des autres parties du programme traitant les différents évènements pouvant intervenir et faisant appel aux fonctions de traitement principales ou aux fonctions utilitaires peut être déduit des explications ci-dessus et du listing figurant dans les figures.

Toute modification à la portée de l'homme de métier fait également partie de l'esprit de l'invention.

Revendications

1. Procédé de retaillage ou de déplacement de fenêtres dans une application "Windows", caractérisé en ce qu'il consiste à interposer entre "windows" (1) et les différentes applications (20, 21, 22) tournant sous "Windows" des filtres (4) interceptant des messages particuliers, à traiter par une application spécifique (5) ces messages et à renvoyer à "Windows" un message neutre "WM_ENTERIDLE" qui ne provoque aucune action de la part de "Windows" et ne bloque pas le fonctionnement des autres applications.

2. Procédé selon la revendication 1, caractérisé en ce que le traitement effectué par l'application spécifique consiste à traiter chacun des évènements particuliers pouvant intervenir et correspondant potentiellement à un évènement de déplacement de la fenêtre ou de retaillage d'une fenêtre, et au cours de ce traitement à faire appel a des fonctions spécifiques permettant d'une part d'initialiser un certain nombre de paramètres "w_Top, w_Right, w_Bottom, w_Caption, FrmCurPos, h_WndCurr" conservés jusqu'à l'occurence d'un évènement ultérieur qui provoquera la fin du traitement.

3. Procédé selon la revendication 2, caractérisé en ce que les paramètres conservés sont constitués par :
   - "h_WndCurr", qui est une variable permettant de savoir qu'une action a été entamée sur une fenêtre ;
   - "b_LoadedIcon", pour mémoriser qu'une icône a été chargée ;
   - "h_WndMenu", qui est la variable permettant de savoir, dans le cas d'une icône, si on est dans une phase de déroulement de menu ou de déplacement d'icône ;
   - "h-OldCursor", qui est la variable permettant de mémoriser un identifieur du curseur actif de Windows avant le lancement de l'application et le remplacement par le curseur specifique de l'application ;

– "w_CXScreen", "w_CYScreen", "w_CXframe", "w_CYMinHeigth", "w_CXMinWidth", qui sont respectivement les variables de coordonnées de la fenêtre en cours de retaillage, de cadre et de largeur mimimum ;
– "Frm_CurPos", qui est la variable de position courante du cadre ;
– "Mse_CurPos", qui est la variable de position courante de la souris ;
– "w_Left", "w_Top", "w_Right", "w_Bottom", "w_Caption", qui sont les variables de direction utilisées dans le calcul des coordonnées de la nouvelle fenêtre et les tests de direction ;
– "Wnd_StartPos", qui est la variable de définition de la position initiale de la fenêtre ;
– "Mse_StartPos", qui est la variable de définition de la position initiale du curseur ;
– "b_Cursor" et "b_LoadedIcon", qui sont des variables booléennes supposées fausses au départ.

4. Procédé selon la revendication 1 ou 3, caractérisé en ce que les filtres sont en nombre de 2.
– un premier WH_GETMESSAGE permettant de recevoir les messages postés par les interruptions hardwate, tels que :
WM_NCLBUTTONDOWN
WM_MOUSEMOVE
WM_KEYDOWN
WM_SYSDEYDOWN
– et un deuxième filtre WH_CALLWNDPROC permettant de filtrer et de recevoir les messages envoyés à la methode, tel que :
WM_SYSCOMMAND
WM_ACTIVATEAPP
WM_NCACTIVATE
WM_ACTIVATE

5. Procédé selon la revendication 4, caractérisé en ce que l'interception d'un message déclenche le traitement d'un programme spécifique à chaque message et fait intervenir les paramètres conservés et des fonctions principales de traitement telles que :
– ABMSInit pour initialiser un traitement ;
– ABMSMove pour effectuer le déplacement de la fenêtre ;
– et ABMSEnd gour terminer un traitement ;
et qui elles-mêmes font appel à des fonctions utilitaires.

6. Procédé selon la revendication 5, caractérisé en ce que la fonction ABMSInit est mise en oeuvre uniquement pour le traitement des messages "WM_BUTTONDOWN" et "SC_SIZE" qui est un message dépendant de "WM_SYSCOMMAND".

7. Procédé selon la revendication 5, caractérisé en ce que la fonction ABMSEnd est mise en oeuvre uniquement pour le traitement des messages "WM_LBUTTONUP" et "VK_ESCAPE", "VK_RETURN" qui dépendent de "WM_SYSKEYDOWN".

8. Procédé selon la revendication 5, caractérisé en ce que la fonction ABMSMove est mise en oeuvre uniquement pour le traitement des messages "WM_MOUSEMOVE, VK_LEFT, VK-UP, VK_RIGHT et VK_DOWN qui sont des messages dépendant de WM_SYSKEYDOWN.

9. Procédé selon la revendication 6, caractérisé en ce que la fonction ABMSInit consiste à approprier les messages souris par l'instruction SETCAPTURE, puis à regarder si la fenêtre est une fenêtre fille d'une fenêtre mère, c'est-à-dire inscrite dans la fenêtre mère pour limiter les mouvements de la fenêtre fille, puis à initialiser les coordonnées initiales de la fenêtre et la position du curseur pour devenir les coordonnées courantes et à initialiser les paramètres caractéristiques de la fenêtre (w-CXframe, w_CXMindWidth, w_CYMinHeigth).

10. Procéde selon la revendication 1 ou 2, caractérisé en ce qu'il comprend les étapes suivantes, de la part de l'application spécifique :
– passage en mode actif des filtres ;
– mémorisation de l'identifieur de la fenêtre h_Wnd, destinataire du message ;
– mémorisation du type d'action résultant de la zone de clic, cette mémorisation se faisant par la fonction HitTest (501) et la fonction ABMSInit (502, 503) ;
– appropriation des messages souris ultérieurs pour le compte de la fenêtre en cause ;
– premier dessin du cadre fantôme autour de la fenêtre par la procédure InvertBlock (5030) ;
– neutralisation du message par remplacement et substitution d'un message "WM_ENTERIDLE" qui n'est pas traité et ne provoque aucune action de la part de Windows.

11. Procédé selon la revendication 10, caractérisé en ce qu'il comprend les étages suivantes :
– calcul des coordonnées finales de la fenêtre par la procédure ABMSComputNewPos, effacement du fantôme par InvertBlock ;
– dessin de la fenêtre en position finale par la fonction ABMSMove (511) ;
– remise à 0 des paramètres de mémorisation par la fonction ABMSEnd (521).

12. Procédé selon la revendication 11, caractérisé en ce qu'il comprend en outre les étapes suivantes :

– abandon de la propriété des messages souris par l'instruction Release Capture ;
– passage du filtre (4) en mode passant.

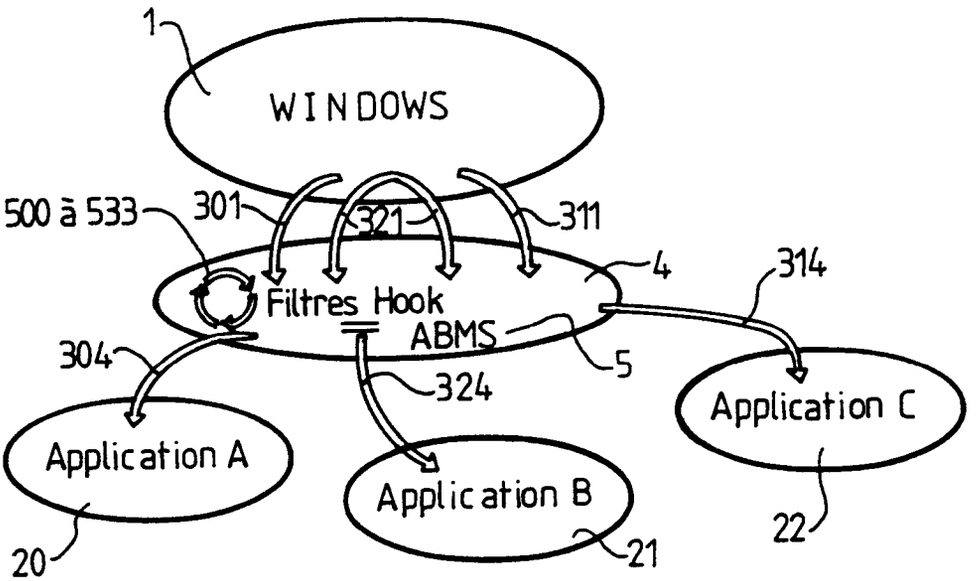
1
WINDOWS
500 à 533
301
321
311
4
314
Filtres Hook
ABMS
5
304
324
Application C
Application A
Application B
20
21
22

FIG.1

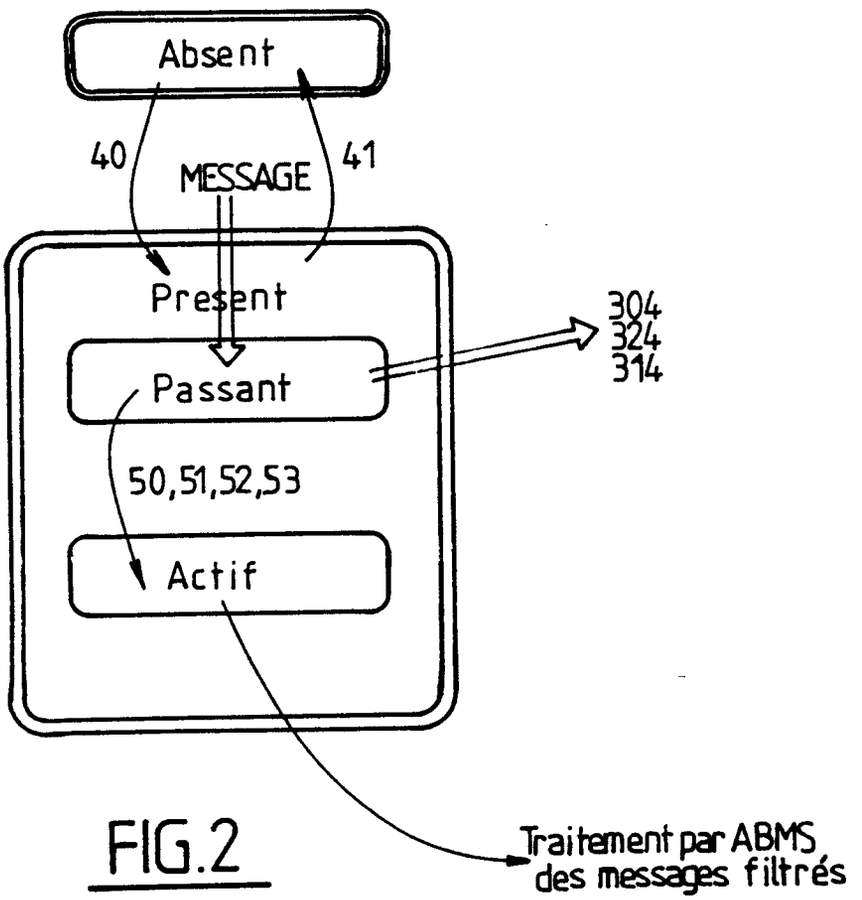
Absent
40
MESSAGE
41
Present
304
324
314
Passant
50,51,52,53
Actif
Traitement par ABMS
des messages filtrés

FIG.2

		Souris	Clavier
Déplacement Fenêtre	Iconic	Differencier debut deplacement et deroulement Menu	
	Non Iconic		
Retaillage Fenêtre	Iconic		
	Non Iconic	○	Detection du sens de retaillage .

FIG.3

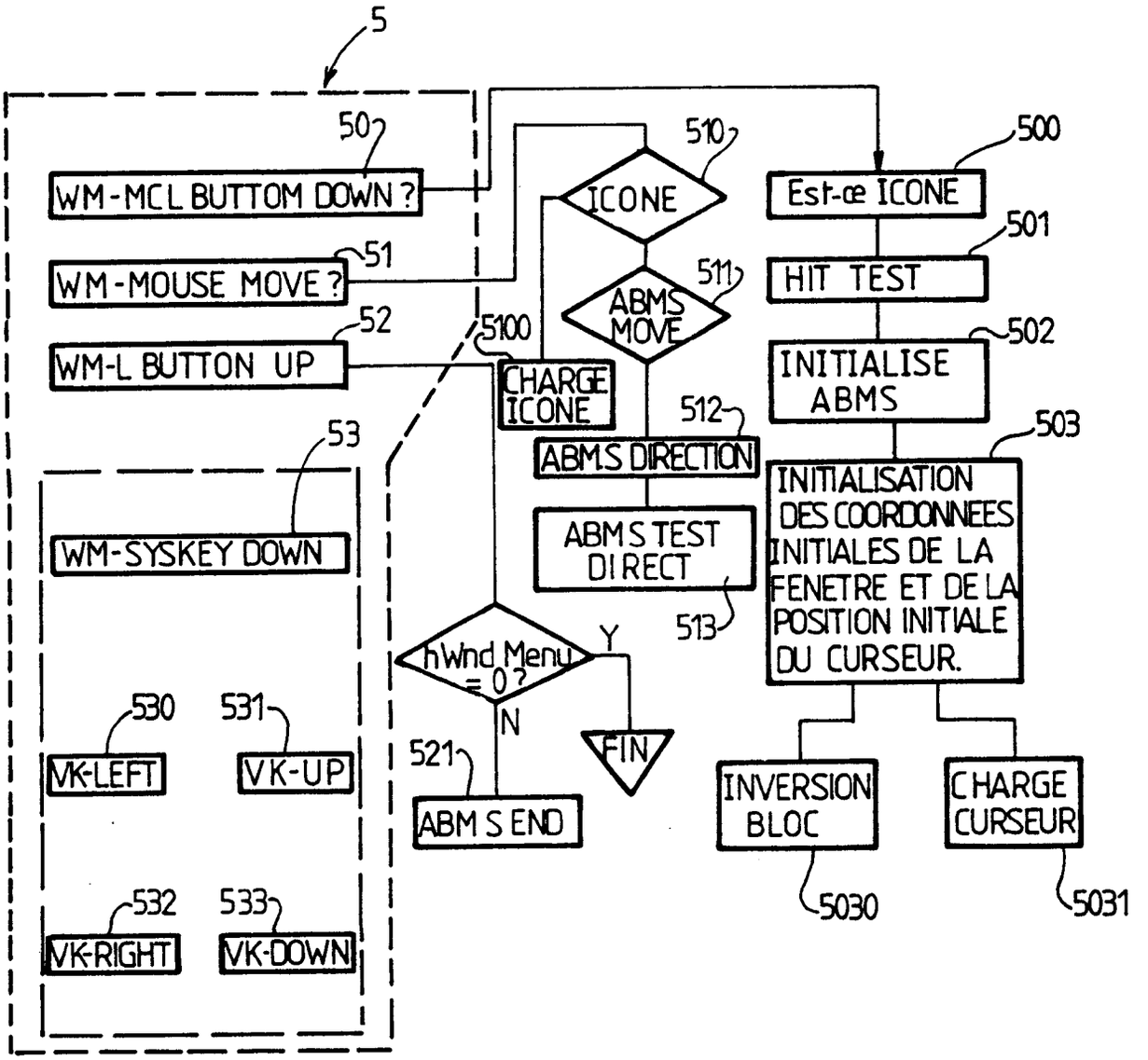
5
50
WM-MCL BUTTOM DOWN ?
51
WM-MOUSE MOVE ?
52
WM-L BUTTON UP
53
WM-SYSKEY DOWN
530
VK-LEFT
531
VK-UP
532
VK-RIGHT
533
VK-DOWN
510
ICONE
511
ABMS MOVE
5100
CHARGE ICONE
512
ABMS DIRECTION
ABMS TEST DIRECT
513
hWnd Menu = 0 ?
Y
N
FIN
521
ABMS END
500
Est-ce ICONE
501
HIT TEST
502
INITIALISE ABMS
503
INITIALISATION DES COORDONNEES INITIALES DE LA FENETRE ET DE LA POSITION INITIALE DU CURSEUR.
INVERSION BLOC
5030
CHARGE CURSEUR
5031

FIG.4

FIG.5

```
/*****************Install and UnInstall Functions************************/
void FAR PASCAL OpenHook(HWND P_hWnd)                                    }
{                                                                        }
    h_WndIcon = P_hWnd;                                                  }
                                                                         }
    w_CXScreen = GetSystemMetrics (SM_CXSCREEN);                         }
    w_CYScreen = GetSystemMetrics (SM_CYSCREEN);                         }
                                                                         }40
    lp_fnOldGetMsgHook =                          400                    }
       (FARPROC)SetWindowsHook(WH_GETMESSAGE, (FARPROC)GetMsgHook);      }
    lp_fnOldCalMsgHook =                                                 }
       (FARPROC)SetWindowsHook(WH_CALLWNDPROC, (FARPROC)CalMsgHook);     }
}                                             401                        }
/*********************************************************************/  }
BOOL FAR PASCAL CloseHook()                                              }
{                                                                        }41
    UnhookWindowsHook(WH_CALLWNDPROC, (FARPROC)CalMsgHook);              }
    return ((BOOL)UnhookWindowsHook(WH_GETMESSAGE, (FARPROC)GetMsgHook));}
}                                                                        }
/*********************************************************************/

/***************************** Hook Functions ****************************/
DWORD FAR PASCAL CalMsgHook(int iCode, WORD wParam, LONG lParam)
{
    MYMSG       Msg;
    POINT       MsePos;

    if (iCode < 0)
        return (DefHookProc(iCode, wParam, lParam, &lp_fnOldGetMsgHook));

    /* The current message is to be processed */                         }
    Msg = * (( LPMYMSG)lParam);                                          }
    switch (Msg.message) {                                               }
       case WM_SYSCOMMAND:                                               }
          switch (Msg.wParam) {                                          }
             case SC_MOVE:                                               }
                w_Caption = 1;                                           }
             case SC_SIZE:                    5401                       }
                GetCursorPos (&MsePos);                                  }
                ABMSInit (Msg.hwnd, D_CURSORMOVESIZEWND, MsePos);  5402  }
                /* Cursor must be set according to the action */         }
                Mse_CurPos.x = Wnd_StartPos.left + (Wnd_StartPos.right/2);}
    540         Mse_CurPos.y = Wnd_StartPos.top+ ((Msg.wParam == SC_MOVE) ?} 54
                                          10 : (Wnd StartPos.bottom/2));  }
                SetCursorPos(Mse_CurPos.x, Mse_CurPos.y);                }
                ((LPMYMSG)lParam)->message = WM_ENTERIDLE;        5403   }
                break;                                                   }
                                          541                            }
             default:                                                    }
                break;                                                   }
             }                                                           }
          break;                                                         }

       case WM_ACTIVATEAPP:                                              }
       case WM_NCACTIVATE:                                               }
       case WM_ACTIVATE:                                                 }
          if (h_WndCurr && (Msg.hwnd != h_WndCurr) && (Msg.wParam != 0))  }
             ((LPMYMSG)lParam)->wParam = 0;                              }
          break;                                                         }
                                                                         }55
       default:                                                          }
          break;                                                         }
       }                                                                 }
    return (DefHookProc(iCode, wParam, lParam, &lp_fnOldCalMsgHook));    }
}                                                                        }
/*********************************************************************/
```

FIG. 5

```
DWORD FAR PASCAL GetMsgHook(int iCode, WORD wParam, LONG lParam)
{
   MSG       Msg;
   POINT     MsePos;

   if (iCode < 0)
      return (DefHookProc(iCode, wParam, lParam, &lp_fnOldGetMsgHook));

   /* The current message is to be processed */
   Msg = *((LPMSG)lParam);
   switch (Msg.message) {
      case WM_NCLBUTTONDOWN:
         if ((h_WndCurr != Msg.hwnd) && (h_WndMenu != Msg.Hwnd)) {
   504      /* No selection and Not sent to init the system menu */
   500      if (IsIconic (Msg.hwnd)) Msg.wParam = HTCAPTION;
   501      if (ABMSHitTest (Msg.wParam)) {
               SetFocus (Msg.hwnd);
   503,502     ABMSInit (Msg.hwnd, 0, MAKEPOINT(Msg.lParam));
               ((LPMSG)lParam)->message = WM_ENTERIDLE;
               }                                              505
            }
         break;                                                          } 50

      case WM_MOUSEMOVE:
         if (h_WndCurr == Msg.hwnd) {
            MsePos = MAKEPOINT(Msg.lParam);
            ClientToScreen(Msg.hwnd, &MsePos);
   510      if (!IsIconic(Msg.hwnd)) {
   515         if (w_Caption+w_Top+w_Left+w_Bottom+w_Right !=0)
   511            ABMSMove (MsePos);
               else
   512            ABMSDirection (ABMSTestDirect (MsePos));
               }                                                       } 51
            else {                                          513
   5111        if (!b_LoadedIcon &&
                  ((MsePos.x != Mse_CurPos.x) || (MsePos.y != Mse_CurPos.y)))
   5100           ABMSLoadIcon (Msg.hwnd);
               }
            ((LPMSG)lParam)->message = WM_ENTERIDLE;
            }                                               514
         break;
```

FIG.6

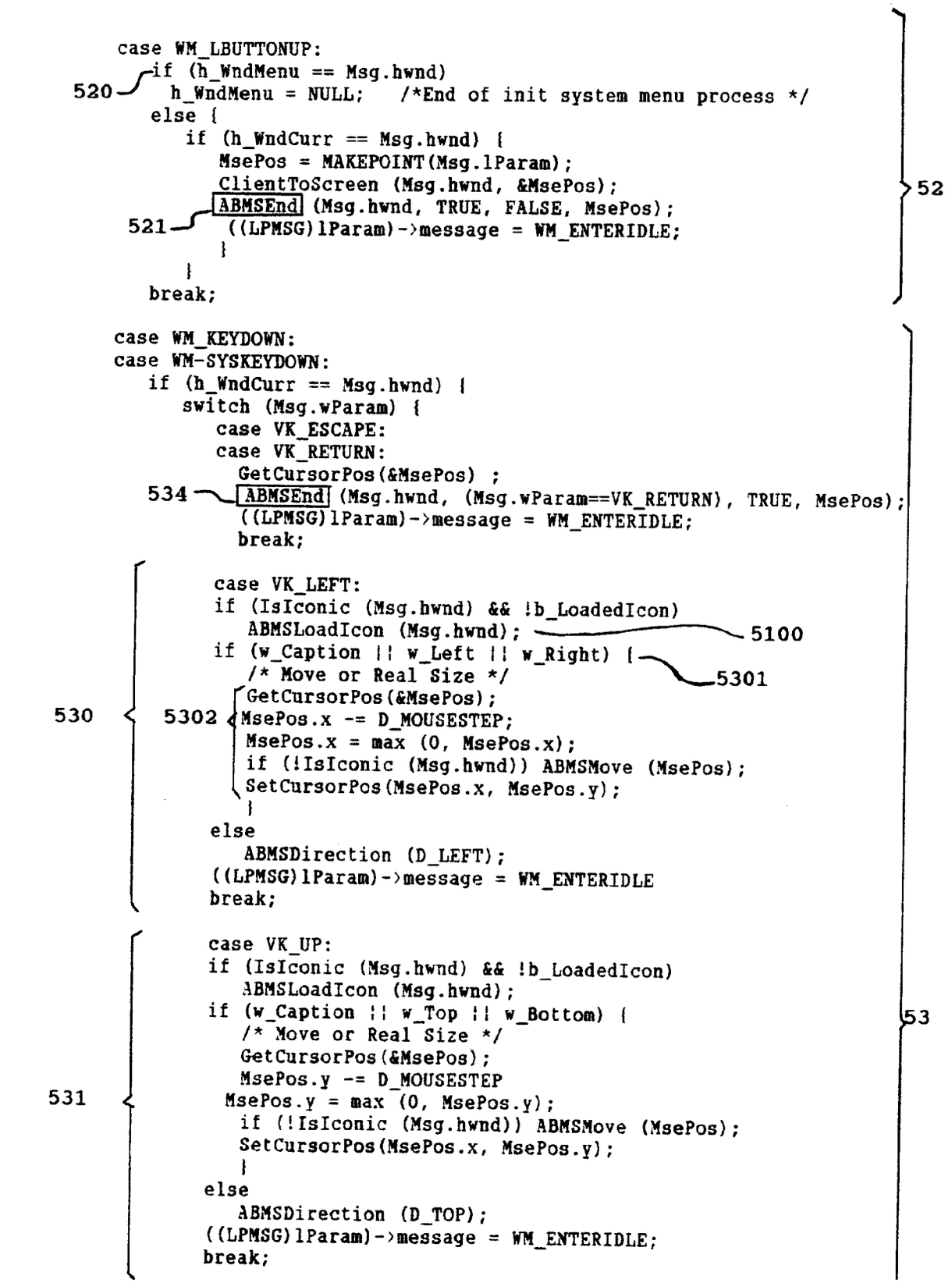
case WM_LBUTTONUP:
if (h_WndMenu == Msg.hwnd)
520
h_WndMenu = NULL; /*End of init system menu process */
else {
if (h_WndCurr == Msg.hwnd) {
MsePos = MAKEPOINT(Msg.lParam);
ClientToScreen (Msg.hwnd, &MsePos);
ABMSEnd (Msg.hwnd, TRUE, FALSE, MsePos);
521
((LPMSG)lParam)->message = WM_ENTERIDLE;
}
}
break;
52
case WM_KEYDOWN:
case WM-SYSKEYDOWN:
if (h_WndCurr == Msg.hwnd) {
switch (Msg.wParam) {
case VK_ESCAPE:
case VK_RETURN:
GetCursorPos(&MsePos) ;
534
ABMSEnd (Msg.hwnd, (Msg.wParam==VK_RETURN), TRUE, MsePos);
((LPMSG)lParam)->message = WM_ENTERIDLE;
break;
case VK_LEFT:
if (IsIconic (Msg.hwnd) && !b_LoadedIcon)
ABMSLoadIcon (Msg.hwnd);
5100
if (w_Caption || w_Left || w_Right) {
5301
/* Move or Real Size */
530
5302
GetCursorPos(&MsePos);
MsePos.x -= D_MOUSESTEP;
MsePos.x = max (0, MsePos.x);
if (!IsIconic (Msg.hwnd)) ABMSMove (MsePos);
SetCursorPos(MsePos.x, MsePos.y);
}
else
ABMSDirection (D_LEFT);
((LPMSG)lParam)->message = WM_ENTERIDLE
break;
case VK_UP:
if (IsIconic (Msg.hwnd) && !b_LoadedIcon)
ABMSLoadIcon (Msg.hwnd);
if (w_Caption || w_Top || w_Bottom) {
/* Move or Real Size */
GetCursorPos(&MsePos);
MsePos.y -= D_MOUSESTEP
531
MsePos.y = max (0, MsePos.y);
if (!IsIconic (Msg.hwnd)) ABMSMove (MsePos);
SetCursorPos(MsePos.x, MsePos.y);
}
else
ABMSDirection (D_TOP);
((LPMSG)lParam)->message = WM_ENTERIDLE;
break;
53

FIG. 6

```
                case VK_RIGHT:
                if (IsIconic (Msg.hwnd) && !b_LoadedIcon)
                   ABMSLoadIcon (Msg.hwnd);
                if (w_Caption || w_Left || w_Right) {
                   /* Move or Real Size */
                   GetCursorPos(&MsePos);
                   MsePos.x += D_MOUSESTEP
532              MsePos.x = min (w_CXScreen, MsePos.x);
                   if (!IsIconic (Msg.hwnd)) ABMSMove (MsePos);
                   SetCursorPos(MsePos.x, MsePos.y);
                   }
                else
                   ABMSDirection (D_RIGHT);
                ((LPMSG)lParam)->message = WM_ENTERIDLE
                break;

                case VK_DOWN:
                if (IsIconic (Msg.hwnd) && !b_LoadedIcon)
                   ABMSLoadIcon (Msg.hwnd);
                if (w_Caption || w_Top || w_Bottom) {
                   /* Move or Real Size */
                   GetCursorPos(&MsePos);
533              MsePos.y += D_MOUSESTEP
                   MsePos.y = min (w_CYScreen, MsePos.y);                    } 53
                   if (!IsIconic (Msg.hwnd)) ABMSMove (MsePos);
                   SetCursorPos(MsePos.x, MsePos.y);
                   }
                else
                   ABMSDirection (D_BOTTOM);
                ((LPMSG)lParam)->message = WM_ENTERIDLE;
                break;
             default:
                break;
             }
          }
       break;

    default:
       break;
    }
 return (DefHookProc(iCode, wParam, lParam, &lp_fnOldGetMsgHook));
 }
 /*************************************************************************/
```

FIG.7

```
/****************** Steps Functions ***********************************/
VOID ABMSInit (HWND P_hWnd, WORD P_wCursor, POINT P_MsePos)
{
   LONG lStyle;
   RECT ClipRect;

   /* Selection begins at this point */
   h_WndCurr = P_hWnd;
   SetCapture(P_hWnd);

   /* If current window is a child one, limits its movement */
   lStyle = GetWindowLong (P_hWnd, GWL_STYLE);
   if (lStyle & WS_CHILD) {
      HWND hParent;
      hParent = GetParent (P_hWnd);
      GetClientRect (hParent, &ClipRect);
      ClientToScreen(hParent, (LPPOINT)&ClipRect);
      ClientToScreen(hParent, (LPPOINT)&(ClipRect.right));
      ClipCursor (&ClipRect);
      }

   /* Initialize the Window Initial Coordinates and the cursor initial*/
   /* position...                                                     */
   GetWindowRect(P_hWnd, &Wnd_StartPos);
   Wnd_StartPos.right -= Wnd_StartPos.left;
   Wnd_StartPos.bottom -= Wnd_StartPos.top;
   Mse_StartPos = P_MsePos;

   /* ... which are also the current one !                            */
   Frm_CurPos = Wnd_StartPos;
   Mse_CurPos = Mse_StartPos;

   /* Initialize the Window characteristics parameters.               */
   w_CXframe = GetSystemMetrics(SM_CXFRAME);
   w_CXMinWidth = GetSystemMetrics(SM_CXMINTRACK);
   w_CYMinHeigth = GetSystemMetrics(SM_CYMINTRACK);

   if (!IsIconic (P_hWnd))
      ABMSInvertBlock(Frm_CurPos, w_CXframe);

   if (P_wCursor) ABMS_LoadCursor (P_wCursor);
}
/***************************************************************************/
```

502

503

FIG.8

```
VOID ABMSEnd (HWND P_hWnd, BOOL P_bNormalEnd, BOOL bRestoreMse,
              POINT P_MsePos)
{
   LONG lStyle;
5211
   if (!IsIconic (P_hWnd)) ABMSInvertBlock(Frm_CurPos, w_CXframe);
   if (P_bNormalEnd)
5212  ABMSComputNewPos (P_MsePos);        5210
   else
      Frm_CurPos = Wnd_StartPos;        5213

   /* If current Window is a child one, converts coord. before move */
   lStyle = GetWindowLong (P_hWnd, GWL_STYLE);
   if (lStyle & WS_CHILD) {
      HWND hParent;
      hParent = GetParent (P_hWnd);
5214  ScreenToClient(hParent, (LPPOINT)&Frm_CurPos);
      ClipCursor (NULL);
      }

   MoveWindow(P_hWnd, Frm_CurPos.left, Frm_CurPos.top,
                      Frm_CurPos.right, Frm_CurPos.bottom, TRUE);

   /* Restore Cursor if needed */
   if (h_OldCursor)
      SetCursor (h_OldCursor);
   else
      if (b_Cursor) SendMessage (h_WndIcon, WM_DDE_ACK, 0, 0L);
   if (bRestoreMse) SetCursorPos(Mse_StartPos.x, Mse_StartPos.y);

   if (IsIconic(P_hWnd)) {
      if (!b_LoadedIcon) {
5215     h_WndMenu = P_hWnd;
         PostMessage(P_hWnd, WM_NCLBUTTONDOWN, HTCAPTION,
                     MAKELONG (P_MsePos.x, P_MsePos.y));
         PostMessage(P_hWnd, WM_LBUTTONUP, 0,
                     MAKELONG (P_MsePos.x, P_MsePos.y));
         }
      else {
         ShowWindow(P_hWnd, SW_SHOWMINIMIZED);
         SetFocus (P_hWnd);
         }
      }

   w_Left = 0;
   w_Top = 0;
   w_Right = 0;
   w_Bottom = 0;
   w_Caption = 0;
5216
   b_Cursor = FALSE;
   h_OldCursor = NULL;
   b_LoadedIcon = FALSE;
   ReleaseCapture();
   h_WndCurr = NULL;
}
/*****************************************************************************/
```

521

FIG.9

```
VOID ABMSMove (POINT P_MsePos)
{
   ABMSInvertBlock(Frm_CurPos, w_CXframe);~5030
   ABMSComputNewPos(P_MsePos);~5210
   ABMSInvertBlock(Frm_CurPos, w_CXframe);~5112
}
/*****************************************************************************/

/************************** Utilities Functions **************************/
void ABMSComputNewPos(POINT P_MsePos)
{
   POINT     MseNewPos;
   RECT      FrmNewPos;

   /* Compute Usable New Cursor Position by including the size limits */
   MseNewPos = P_MsePos;
   MseNewPos.x = (w_Left+w_Right) ?
    (w_Left * min(MseNewPos.x, Frm_CurPos.left+Frm_CurPos.right-w_CXMinWidth)
     + w_Right * max(MseNewPos.x, Frm_CurPos.left+w_CXMinWidth)) :
     MseNewPos.x;
   MseNewPos.y = (w_Top+w_Bottom) ?
     (w_Top * min (MseNewPos.y, Frm_CurPos.top+Frm_CurPos.bottom-w_CYMinHeigth)
     + w_Bottom * max (MseNewPos.y, Frm_CurPos.top+w_CYMinHeigth)) :
     MseNewPos.y ;

   /* Compute New Frame Position and Size */
   FrmNewPos.left = (w_Left+w_Caption) * MseNewPos.x
                  + (1-w_Left) *Frm_CurPos.left - w_Caption * Mse_CurPos.x;
   FrmNewPos.top = (w_Top+w_Caption) * MseNewPos.y
                 + (1-w_Top) * Frm_CurPos.top - w_Caption * Mse_CurPos.y;
   FrmNewPos.right = (w_Right-w_Left) * (MseNewPos.x - Frm_CurPos.left)
                   + (1-w_Right) * Frm_CurPos.right;
   FrmNewPos.bottom = (w_Bottom-w_Top) * (MseNewPos.y - Frm_CurPos.top)
                    + (1-w_Bottom) * Frm_CurPos.bottom;

   /* Save the New Current Values for Mouse Position and Frame Coord.*/
   Mse_CurPos = P_MsePos;
   Frm_CurPos = FrmNewPos;
}
/*****************************************************************************/
```

511

5210

FIG. 10

```
VOID ABMSDirection (WORD P_wDirect)
{
   WORD wCursor = 0;

   switch (P_wDirect) {
      case D_LEFT:
         w_Left = 1;
         Mse_CurPos.x = Frm_CurPos.left;
         wCursor = (w_Top) ? D_TTOBDBLEARROW :
                           ((w_Bottom) ? D_BTOTDBLEARROW :
                                         D_HORZDBLEARROW);
         break;

      case D_TOP:
         w_Top = 1;
         Mse_CurPos.y = Frm_CurPos.top;
         wCursor = (w_Left) ? D_TTOBDBLEARROW :
                           ((w_Right) ? D_BTOTDBLEARROW :
                                        D_VERTDBLEARROW);
         break;

      case D_RIGHT:
         w_Right = 1;
         Mse_CurPos.x = Frm_CurPos.left+Frm_CurPos.right;
         wCursor = (w_Bottom) ? D_TTOBDBLEARROW :
                              ((w_Top) ? D_BTOTDBLEARROW :
                                         D_HORZDBLEARROW);
         break;

      case D_BOTTOM:
         w_Bottom = 1;
         Mse_CurPos.y = Frm_CurPos.top+Frm_CurPos.bottom;
         wCursor = (w_Right) ? D_TTOBDBLEARROW :
                           ((w_Left) ? D_BTOTDBLEARROW :
                                       D_VERTDBLEARROW);
         break;

      default :
         return;
         break;
      }

   SetCursorPos(Mse_CurPos.x, Mse_CurPos.y);
   SendMessage (h_WndIcon, WM_DDE_EXECUTE, wCursor,0L);
   b_Cursor = TRUE;
}
/*****************************************************************************/
```

5120

512

FIG. 11

```
BOOL ABMSHitTest (WORD P_wHit) {
   switch (P_wHit) {
      case HTTOP:
         w_Top = 1;break;
      case HTTOPRIGHT:
         w_Top = 1;w_Right = 1;break;
      case HTRIGHT:
         w_Right = 1;break;
      case HTBOTTOMRIGHT:
         w_Bottom = 1;w_Right = 1;break;
      case HTBOTTOM:
         w_Bottom = 1;break;
      case HTBOTTOMLEFT:
         w_Bottom = 1;w_Left = 1;break;
      case HTLEFT:
         w_Left = 1;break;
      case HTTOPLEFT:
         w_Top = 1;w_Left = 1;break;
      case HTCAPTION:
         w_Caption = 1;break;
      default:
         return FALSE;
         break;
      }
   return TRUE;
}
```

501

```
/*****************************************************************************/

void ABMSInvertBlock(RECT P_RecFrm,WORD P_Width)
{
   HDC  hDC;

   hDC = CreateDC("DISPLAY", NULL, NULL, NULL);
   PatBlt(hDC, P_RecFrm.left, P_RecFrm.top,
               P_RecFrm.right-P_Width, P_Width, DSTINVERT);
   PatBlt(hDC, P_RecFrm.left+P_RecFrm.right-P_Width, P_RecFrm.top,
               P_Width, P_RecFrm.bottom-P_Width, DSTINVERT);
   PatBlt(hDC, P_RecFrm.left+P_Width, P_RecFrm.top+P_RecFrm.bottom-P_Width,
               P_RecFrm.right-P_Width, P_Width, DSTINVERT);
   PatBlt(hDC, P_RecFrm.left, P_RecFrm.top+P_Width,
               P_Width, P_RecFrm.bottom-P_Width, DSTINVERT);
   DeleteDC(hDC);
}
/*****************************************************************************/
```

5030

```
VOID ABMSLoadCursor (WORD P_wCursor)
{
   SendMessage (h_WndIcon, WM_DDE_EXECUTE, P_wCursor, 0L);
   b_Cursor = TRUE;
```

5031

FIG. 12

```
}
/*************************************************************************/

VOID ABMSLoadIcon (HWND P_hWnd)
{
   HCURSOR   hIcon;

   ShowWindow(P_hWnd, SW_HIDE);
   SetFocus (P_hWnd);
   if (hIcon = (HCURSOR)GetClassWord(P_hWnd, GCW_HICON))
      h_OldCursor = SetCursor(hIcon);
   else
      ABMSLoadCursor (D_CURSORMOVEICON);
   b_LoadedIcon = TRUE;
}
/*************************************************************************/

WORD ABMSTestDirect (POINT P_MsePos)
{
   if (P_MsePos.x <= Frm_CurPos.left) return (D_LEFT);
   if (P_MsePos.x >= (Frm_CurPos.left+Frm_CurPos.right)) return (D_RIGHT);
   if (P_MsePos.y <= Frm_CurPos.top) return (D_TOP);
   if (P_MsePos.y >= (Frm_CurPos.top+Frm_CurPos.bottom)) return (D_BOTTOM);
   return (0);
}
/*************************************************************************/
```

5100

513

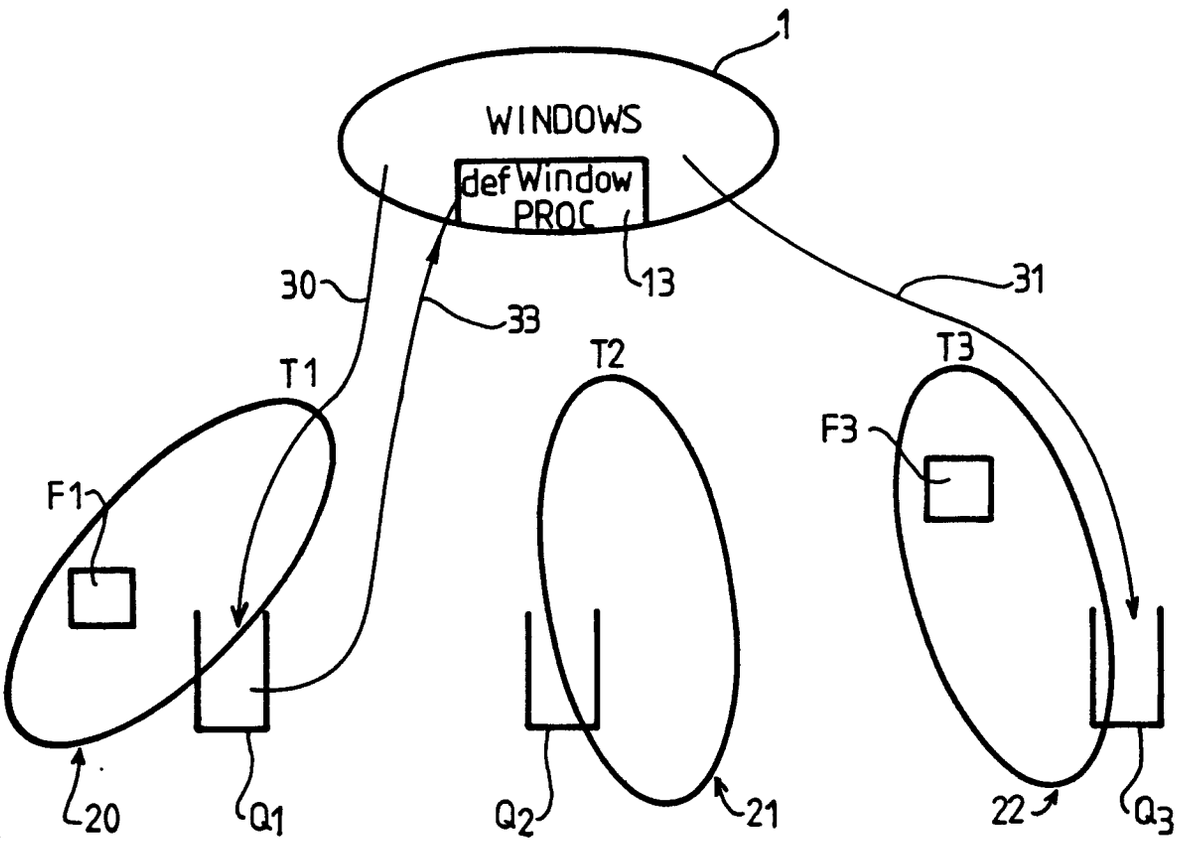
1
WINDOWS
defWindow
PROC
13
30
33
31
T1
T2
T3
F1
F3
20
Q1
Q2
21
22
Q3

FIG.13

Office européen des brevets

RAPPORT PARTIEL DE RECHERCHE EUROPEENNE

qui selon la règle 45 de la Convention sur le brevet européen est considéré, aux fins de la procédure ultérieure, comme le rapport de recherche européenne

Numéro de la demande: EP 91 40 2070

DOCUMENTS CONSIDERES COMME PERTINENTS

Catégorie	Citation du document avec indication, en cas de besoin, des parties pertinentes	Revendication concernée	CLASSEMENT DE LA DEMANDE (Int. Cl.5)
A	PROC. OF THE 1st INT. CONF. ON COMPUTER WORKSTATIONS, San novembre 1985, San Jose, US, pages 12-21; M.J. GOODFELLOW: "WHIM, The window handler and input manager" * Sections 1, 3.1, 3.2; note en bas de page 15 * --	1	G 09 G 1/00 G 06 F 3/14
A	COMPUTER, vol. 19, no. 9, septembre 1986, Long Beach, US, pages 57-67; B.A. MYERS: "A complete and efficient implementation of covered windows" * page 65, colonne du milieu, lignes 14-32; page 66, colonne de gauche, ligne 20 * ----------	1	DOMAINES TECHNIQUES RECHERCHES (Int. Cl.5) G 09 G G 06 F

RECHERCHE INCOMPLETE

La division de la recherche estime que la présente demande de brevet européen n'est pas conforme aux dispositions de la Convention sur le brevet européen au point qu'une recherche significative sur l'état de la technique ne peut être effectuée au regard d'une partie des revendications.

Revendications ayant fait l'objet de recherches complètes:

Revendications ayant fait l'objet de recherches incomplètes: 1

Revendications n'ayant pas fait l'objet de recherches: 2 à 12

Raison pour la limitation de la recherche:

voir annexe -C-

Lieu de la recherche	Date d'achèvement de la recherche	Examinateur
LA HAYE	19 septembre 1991	WILTINK J.G.

CATEGORIE DES DOCUMENTS CITES

X : particulièrement pertinent à lui seul
Y : particulièrement pertinent en combinaison avec un autre document de la même catégorie
A : arrière-plan technologique
O : divulgation non-écrite
P : document intercalaire

T : théorie ou principe à la base de l'invention
E : document de brevet antérieur, mais publié à la date de dépôt ou après cette date
D : cité dans la demande
L : cité pour d'autres raisons

& : membre de la même famille, document correspondant

OEB Form 1505.1 03.82

EP 91 40 2070 -C-

La recherche incomplète pour la revendication 1 a été faite pour des "procédés de retaillage ou de déplacement de fenêtres" dans le contexte du multiprogrammation. Les aspects specifiques du filtrage et du renvoi des messages n'ont pas été pris en compte.

Une telle demande, notamment les revendications 2 à 12, qui revendiquent explicitement certains éléments d'un programme, comme "w-Top","b-loaded Icon", ne me semble pas admissible sous le système Européen, vu l'article 52(2)c de la Convention.