ROYAUME DE BELGIQUE

MINISTERE DES AFFAIRES ECONOMIQUES

BREVET D'INVENTION

NUMERO DE PUBLICATION : 1008621A3

NUMERO DE DEPOT : 09401092

Classif. Internat. : G06F

Date de délivrance le : 04 Juin 1996

Le Ministre des Affaires Economiques,

Vu la Convention de Paris du 20 Mars 1883 pour la Protection de la propriété industrielle;

Vu la loi du 28 Mars 1984 sur les brevets d'invention, notamment l'article 22;

Vu l' arrêté royal du 2 Décembre 1986 relatif à la demande, à la délivrance et au maintien en vigueur des brevets d'invention, notamment l'article 28;

Vu le procès verbal dressé le 01 Décembre 1994 à 14H15 à l 'Office de la Propriété Industrielle

ARRETE:

ARTICLE 1.- Il est délivré à : MARK BLUNDELL & ASSOCIATES, INC. Byram Ridge Road 78, ARMONK, NEW YORK 10504(ETATS-UNIS D'AMERIQUE)

représenté(e)(s) par : VAN MALDEREN Joëlle, OFFICE VAN MALDEREN, Place Reine Fabiola 6/1 - B 1083 BRUXELLES.

un brevet d' invention d' une durée de 20 ans, sous réserve du paiement des taxes annuelles, pour : PROCEDURE ET PROCEDE DE COMMUNICATION ENTRE MACHINES ET PROCEDE GENERALISE DE PREPARATION DE PROGRAMMES AFFERENTS.

INVENTEUR(S) : Brann John E.T., Bleeker Street 375, New York, New York 10014 (US)

PRIORITE(S) 02.12.93 US USA 161229

ARTICLE 2.- Ce brevet est délivré sans examen préalable de la brevetabilité de l'invention, sans garantie du mérite de l'invention ou de l'exactitude de la description de celle-ci et aux risques et périls du(des) demandeurs(s).

Bruxelles, le 04 Juin 1996
PAR DELEGATION SPECIALE :

L. WUYTS
CONSEILLER

Procédure et procédé de communication entre machines et procédé généralisé de préparation de programmes afférents

Arrière-plan de l'invention - Domaine d'application

La présente invention concerne le domaine de la construction de programmes informatiques et, en particulier, la construction de programmes de traduction de données. Plus particulièrement, l'invention vise un procédé de liaison retardée pour offrir un système efficace d'assemblage et de traduction d'informations d'un ordinateur à un autre.

Arrière-plan de l'invention - Description de la technique antérieure

L'histoire des systèmes informatiques et des programmes utilisés dans ceux-ci s'est rapidement développée au cours des 35 dernières années. Les premiers procédés à être automatisés étaient des tâches de traitement de données séquentielles à grande échelle, telles que la production de feuilles de paye. Au fur et à mesure que chaque problème a été abordé et résolu, on a créé un ensemble de programmes machine. La nature de l'industrie est cependant telle que l'on préfère plutôt utiliser de nouveaux programmes qu'améliorer les solutions existantes. Cela a entraîné la construction de nouveaux systèmes qui ne sont pas apparentés à ceux qui existent déjà. Au fil du temps, le nombre de systèmes indépendants a augmenté énormément. On a accordé peu d'attention à la coordination des données entre les divers programmes

concurrents.

L'énorme quantité de programmes concurrents a mené au développement de techniques de construction de logiciels pour permettre la reconstruction de systèmes de manière plus efficace. Le développement de telles techniques a entraîné l'apparition de langages de quatrième génération (4GL) au début des années 80. Bien que l'on ait largement prévu que les 4GL rendraient le remplacement de tels systèmes aisée et rapide, en permettant de remplacer de manière pratique les grands systèmes aussi fréquemment que tous les quatre ou cinq ans, cela ne s'est pas produit.

Lorsqu'on améliore des programmes, la totalité du programme exécutable doit être remplacé. Cela est une conséquence du procédé de création de programmes qui prend de nombreuses pièces d'un code de source manuel, les compile chacune en code object et construit ensuite le programme exécutable unique à partir de ces objets. Par suite, tout changement d'une pièce quelconque d'un code de source crée une modification importante dans le programme exécutable, ce qui peut donner lieu à des effets secondaires. Après réécriture, le programme doit être soumis à des tests de régression dans lesquels la totalité du programme est retestée pour détecter l'un quelconque de ces effets secondaires. Une autre difficulté qui apparaît est celle du transfert d'une version d'un programme à un autre programme. Cela implique une durée d'immobilisation notable et une reformation du personnel. Il faut ajouter à cela des retards importants pour les essais. Une tentative visant à améliorer ce problème a été l'usage d'outils de génie logiciel assistés par ordinateur tels que des générateurs de codes. Ces techniques permettent, avec

divers degrés de succès, d'accélérer la création de programmes. Cependant, elles ne s'attaquent pas à la difficulté principale des nombreux stades qui doivent être effectués pour améliorer un programme ou en créer un nouveau. Les stades de préparation d'un programme ou de révision du programme dans la technique antérieure peuvent être observés dans la Fig. 1 dans laquelle on passe de "Demande de Changement" à "Concevoir Changements du Logiciel", puis à "Conception". "Programme original" et "Conception" passent alors à "changer programme", puis à "Programme Changé", à "Test du Système", à "Nouveau Programme Accepté", à "Remplacer Programme Original" et enfin à "Nouveau Programme".

Une autre approche de la création de programmes est désigné de manière générale par Orientation d'Objet ("OO"). La théorie générale qui se cache derrière cette méthode de préparation de programmes est le fait que, pour tout type de système informatique, on utilise dans de nombreux cas un ensemble d'opérations non triviales et d'informations associées. Par exemple, dans un système informatique qui automatise certaines opérations bancaires, il y a de fréquentes demandes de traitement des quantités de monnaie. Ces procédés ne font pas partie du fonctionnement de base d'un ordinateur. Ils sont composés de diverses opérations de calcul qui forment une partie commune de nombreux algorithmes plus grands. Dans un environnement OO, ces algorithmes partiels et leurs structures de données associées sont connues sous le nom d'"objets". Cette technique est une extension de la pratique bien connue de création de sous programmes pour des tâches qui se répètent couramment. Les langages de

programmation OO comprennent des structures permettant de rendre cette approche d'une division des programmes plus aisée et plus puissante que dans les langages antérieurs. L'effet global des techniques OO est similaire à celui des générateurs de codes: il s'agit d'améliorer le stade de création de "programmes originaux" (Fig. 1).

Les difficultés dues aux essais du système et au remplacement du programme sont dues à la différence entre le code de source informatique et les images exécutables. Le code de source est créé par des programmeurs ou générateurs de codes. Il se trouve normalement dans certains langages raisonnablement lisibles par l'utilisateur (par exemple, les langages COBOL, FORTRAN, C, PROLOG, etc.). Chacun de ces langages a des règles de grammaire et de syntaxe très formelles. Le code de source est utilisé pour transcrire des algorithmes d'un modèle sous une forme qui peut être interprétée par un compilateur. Les compilateurs sont des programmes qui convertissent le code de source en forme intermédiaire, le code objet. Le code objet est une forme intermédiaire entre le code de source lisible par les utilisateurs et l'image exécutable qui peut réellement tourner l'ordinateur. Le dernier stade est l'exécution d'un programme appelé éditeur de liens qui prend une ou plusieurs pièces du code objet et la (les) convertit en une forme qui peut être exécutée par un ordinateur. L'organigramme de ce procédé de préparation de programmes de la technique antérieure est représenté dans la Fig. 2. L'image exécutable est la seule forme du programme qui tourne sur l'ordinateur. Elle seule détermine le comportement du programme. Le code de source est la seule forme manipulée par des programmeurs ou générateurs

de codes. Les effets secondaires, dont il a été question ci-dessus, peuvent apparaître du fait que l'image exécutable d'un programme quelconque est très complexe. Il est toujours possible qu'apparaissent dans l'image exécutable des effets secondaires non prévus qui n'étaient pas envisagés par le programmeur ou le générateur de code du code de source.

Les programmes globaux qui se posent dans le remplacement de système et qui sont dus à la logistique de conversion des données contenues dans ces systèmes mettant en oeuvre le nouveau logiciel et la rééducation des utilisateurs signifient que ces systèmes continueront à exister pendant une période de temps considérable. Cela implique que ces systèmes doivent être utilisés dans des systèmes à réseau multi-utilisateurs pour que les données que ces systèmes contiennent puissent être utilisées par des systèmes développés par la suite.

Une autre difficulté encore réside dans l'interaction entre différents systèmes. Très souvent, la connexion entre de multiples systèmes est réalisée à la main. Par exemple, des enregistrements imprimés provenant d'un système informatique sont ensuite saisis manuellement dans un deuxième système. C'est lent, fastidieux et, par suite, inefficace et coûteux. Une autre approche bien connue consiste à placer un nouveau programme entre les systèmes existants pour agir comme intermédiaire. Dans l'un ou l'autre cas, une programmation doit être faite.

RESUME DE L'INVENTION

Un objet de l'invention réside dans un procédé informatisé de traduction efficace de messages de source en messages de cible.

Un autre objet encore de l'invention vise un programme machine dans lequel on peut effectuer des changements au programme sans devoir réécrire le code de source entier du programme.

Un autre objet encore de l'invention vise un procédé de création de programmes par stockage et appel d'éléments de programmation de base et combinaison de ces éléments pour usage au moment de l'exécution du programme.

Les objets de la présente invention visent un procédé de programmation machine à liaison retardée pour agir sur des messages de source lors du réglage d'un ordinateur du type ayant une mémoire temporaire et dans lequel le programme traduit un message de source prédéterminé en un message de cible prédéterminé, le programme de liaison retardée comprenant les stades suivants :

(a) on utilise un programme de commande,

(b) on délivre les messages de source au programme de commande,

(c) on utilise une base de données,

(d) on stocke des instructions et des données dans la base de données,

(e) on amène le programme de commande à accéder à la base de données,

(f) on amène le programme de commande à répondre à des instructions et des données stockées dans la base de données,

(g) on utilise au moins un programme de procédure ou sous-programme de traduction,

(h) on amène le programme de commande, sensible à des instructions stockées dans la base de données, à

appeler le programme de procédure de traduction,

(i) on amène le programme de procédure de traduction, en réponse à des instructions de la base de données, à traduire au moins une partie du message reçu, et

(j) on limite le programme de commande à répondre au message de source, à accéder à la base de données et à exécuter le programme de procédure de traduction en réponse à des instructions et des données stockées dans la base de données.

Une autre forme de réalisation de l'invention vise une procédure de programmation d'ordinateur à liaison retardée pour créer des programmes dans laquelle il est délivré au programme d'ordinateur à liaison retardée des stimuli de source dans lesquels chaque stimulus est du type comprenant un événement d'un ensemble d'un nombre prédéterminé d'événements et une structure d'un ensemble prédéterminé de structures de données, et dans laquelle le programme machine à liaison retardée est du type qui peut appeler au moins un sous- programme (c'est-à-dire, un "programme de procédure d'application"), la procédure du programme machine à liaison retardée comprenant les stades suivants :

(a) on utilise un programme de commande,

(b) on délivre les stimuli de source au programme de commande,

(c) on utilise une base de données,

(d) on stocke des instructions, des données et les structures de stimuli de source quelconques sur lesquelles on doit intervenir dans la base de données,

(e) on amène le programme de commande à accéder à la base de données,

(f) on amène le programme de commande à répondre à des instructions et des données stockées dans la base de données,

(g) on amène le programme de commande, en réponse aux stimuli et à des instructions et des données de la base de données, à être à même d'appeler au moins un programme du procédure d'application, et

(h) on limite le programme de commande à répondre au stimulus de source, à accéder à la base de données et à répondre à des instructions et des données stockées dans la base de données.

Dans une autre forme de réalisation encore de l'invention il est prévu une procédure d'écriture d'un programme de traduction pour traduire un message de source en un message de cible comprenant les stades suivants :

(a) on identifie les caractéristiques et les paramètres du message de cible,

(b) on identifie les caractéristiques du message de source,

(c) on en tire les conditions pour traduire le message de source en message de cible,

(d) on met en oeuvre une base de données,

(e) on stocke dans la base de données des données et des instructions correspondant aux stades nécessaire pour traduire le message de source en message de cible,

(f) on transcrit un programme de procédure de traduction pour chaque transformation de chaque type d'élément du message de source en chaque type d'élément du message de cible,

(g) on stocke dans la base de données une référence aux programmes de procédure de traduction,

(h) on stocke dans la base de données des instructions identifiant les programmes de procédure de traduction qui doivent être appelés,

(i) on limite le programme de commande de manière qu'il soit commandé à tous moments par les instructions et les données de la base de données, et

(j) on amène le programme de commande à obtenir de la base de données, à la réception d'un message de source, les instructions de la base de données permettant de diriger le programme de commande et d'appeler des sous-programmes de procédure de traduction prédéterminés pour traduire le message de source en message de cible.

Dans une autre forme de réalisation, il est prévu un procédé d'écriture d'un programme d'application comprenant les stades suivants :

(a) on établit la liste de tous les messages de stimuli que le programme d'application doit recevoir,

(b) on établit une liste de chaque opération que chaque programme de procédure de traduction d'application doit effectuer et la séquence des opérations,

(c) on met en oeuvre une base de données,

(d) on stocke les stimuli dans une table de la base de données,

(e) on stocke l'instruction pour exécuter le programme de procédure d'application dans la base de données sous la forme d'un message comprenant une référence au programme de procédure d'application et une référence aux paramètres que le programme de procédure d'application utilisera,

(f) on détermine chaque stimulus à recevoir par le

programme d'application et on désigne les stimuli comme message de source,

(g) on détermine les structures paramétriques pour chacun des programmes de procédure d'application à exécuter et on désigne chacun comme message de cible,

(h) on identifie les caractéristiques et paramètres de chaque message de cible,

(i) on identifie les caractéristiques du message de source,

(j) on en tire les conditions pour traduire le message de source en message de cible,

(k) on stocke dans la base de données des données et des instructions correspondant aux stades nécessaires pour traduire le message de source en message de cible,

(l) on obtient un programme de procédure d'application pour chaque stade d'application,

(m) on stocke des instructions dans la base de données identifiant les programmes de procédure d'application à appeler,

(n) on amène des instructions de la base de données à faire en sorte que le programme de commande appelle les programmes de procédure d'application et crée les paramètres associés à chaque programme de procédure d'application à partir des éléments du stimulus de source,

(o) on limite le programme de commande de manière qu'il soit commandé à tout instant par les instructions et les données de la base de données, et

(p) on amène le programme de commande à appeler dans la base de données, à la réception du message de source, les instructions de la base de données pour diriger le programme de commande et appeler des programmes de

procédure d'application prédéterminés pour traduire le message de source en message de cible et délivrer le message de cible en tant qu'au moins une partie du programme complet.

BREVE DESCRIPTION DES DESSINS

La présente invention sera à présent expliquée plus en détail en se référant aux dessins ci-annexés dans lesquels:

la Fig. 1 est un organigramme des stades nécessaires pour préparer un programme de la technique antérieure,

la Fig. 2 est un organigramme des stades de préparation de programmes,

la Fig. 3 est un diagramme représentant la décomposition d'un programme typique,

la Fig. 4 est un diagramme représentant un programme de traduction,

la Fig. est un diagramme d'un programme de procédure,

la Fig. 6 est un diagramme de la structure de données d'un programme de procédure, et

la Fig. 7 est un diagramme représentant des demandes et des réponses à des demandes d'un programme de commande à une procédure.

DESCRIPTION DES FORMES DE REALISATION PREFEREES

Il existe un besoin d'un mécanisme susceptible de favoriser l'intégration rapide de systèmes informatiques et un deuxième besoin d'une voie plus rapide de développement et de changement de systèmes. Une seule solution est à même de donner une réponse à ces deux problèmes.

Le temps nécessaire soit à la préparation d'un

nouveau programme, soit à la réécriture d'un programme peut être prévu avec précision par un concept appelé "temps de liaison". Pour analyser des opérations de programmes informatiques, on utilise le concept de temps de liaison. Le temps de liaison décrit le point de la création d'un programme où son comportement est fixé. Par exemple si un programme contient la ligne :

x := 3 + 4

(ce qui signifie que l'on assigne à une variable x la somme de 3 et 4), la valeur de x créée en exécutant la relation sera toujours de 7. Le comportement en "temps de liaison" de cette instruction est donc le moment auquel le programme est compilé. Ce sera le temps de liaison qui se situe le plus tôt possible.

On considère à présent une autre instruction :

x := a + b

(ce qui signifie "donner à une variable x la valeur de la somme des variables a et b").

En outre, placer l'instruction dans un sous-programme :

sous-programme (a, b)

x := a y + b,

retour x.

le comportement de cette instruction est tel que la valeur de x ne peut pas à présent être déterminée à partir de cet algorithme du fait qu'elle dépend des variables a et b. Les variables a et b font partie du programme qui "appelle" le sous-programme. Cette relation appel/réponse n'est pas déterminée au moment de la compilation. Elle est déterminée lorsque le programme est lié sous une forme exécutable. Par suite, le "temps de

liaison" s'est redéplacé vers le temps "d'édition de liens".

Comme exemple final, on imagine un programme qui contient l'instruction :

x := a + b.

Cette fois, le programme est conçu de telle manière que les valeurs de a et b soient fournies par un utilisateur assis au terminal. Dans ce cas, le comportement du programme n'est pas décidé jusqu'à ce qu'il soit lancé. Le "temps de liaison" est à présent le "temps d'exécution". Ce résultat est utile étant donné que le programme est à même de réaliser de manière correcte des actions qui n'étaient pas explicitement envisagées par le concepteur. Toute personne concevant un tel programme d'addition ne passerait pas par le procédé de conception explicite d'une combinaison bizarre éventuelle de nombres (a, b) qui pourraient être ajoutés l'un à l'autre, mais le programme ajoutera correctement l'un à l'autre deux tels nombres quelconques.

Les programmes avec des "temps de liaison" retardés ont cet avantage qu'ils n'ont pas besoin d'être recompilés ou reliés pour offrir de nouvelles fonctions. La mise en oeuvre de changements de programme et de nouveaux programmes, en utilisant de tels procédés de programmation, réduisent la phase d'essai du système au seul essai nécessaire aux nouvelles fonctions. Les anciennes fonctions ne seraient pas affectées du fait que l'image exécutable n'a pas été altérée de manière quelconque. Comme il n'y a pas de changement de l'image exécutable, il n'est pas nécessaire d'être dans l'obligation de la remplacer dans le système en cours de fonctionnement.

Tous les programmes informatiques expriment des

algorithmes. Ces algorithmes comprennent une séquence d'instructions ou de fonctions simples. Mais, en conséquence, n'importe quel programme informatique complexe peut être décrit en le décomposant en ses éléments apparentés (Fig. 3). La nature des instructions n'est pas définie, mais elles sont généralement prises comme une unité atomique de traitement appropriée à la tâche à effectuer. En effet, dans la construction de systèmes informatiques, la valeur, le forme des "instructions" est habituellement définie par la tâche à effectuer et ensuite en utilisant un langage de programmation approprié. Il est bien connu que ces fonctions sont le mieux mises en oeuvre sous la forme de sous-programmes qui peuvent être réutilisés lors de besoins d'instructions similaires.

Une caractéristique des programmes de liaison retardée est qu'ils nécessitent des informations lorsqu'ils tournent, informations susceptibles de leur dire comment réagir à un stimulus particulier. Dans la description de la liaison en temps d'exécution, les informations étaient prévues par un utilisateur assis au terminal. Il est clair que, dans la plupart des cas, une telle situation ne sera pas acceptable. Un autre mécanisme doit être utilisé pour fournir les informations nécessaires. Un tel mécanisme consiste à maintenir les données de configuration sous la forme d'une base de données. Cette base de données peut contenir les informations utilisées pour configurer un programme de liaison retardée pour un ensemble de circonstances. Les changements de la base de données de configuration modifient le comportement du programme.

La base de données contient des références aux stades de traitement à réaliser pour un stimulus

particulier. Comme la définition d'un algorithme se fait en termes de tels stades de traitement, toute opération qui peut être effectuée par voie algorithmique peut être décrite dans le programme de liaison retardée, configuré par une base de données, qui contient un code de programme préexistant approprié et peut effectuer n'importe quelle procédure que l'ordinateur est à même de réaliser. En conséquence, un programme de liaison retardée peut effectuer n'importe quelle tâche qu'un programme construit à cet effet peut réaliser.

La tâche de connexion conjointe d'un certain nombre de systèmes informatiques peut être réduite à de nombreuses tâches séparées de connexion de paires de systèmes. Le connecteur doit compléter trois tâches séparées. Ces tâches sont les suivantes :

1. Etablir que les données provenant du système d'émission a un sens pour le récepteur,

2. Etablir un moyen de faire passer des informations électroniques lisibles par un ordinateur du premier système au deuxième système, et

3. Restructurer les informations sous une forme acceptable pour le récepteur.

La première tâche est réalisée par examen des systèmes concernés et la deuxième par n'importe quel protocole électronique approprié déterminé par la nature des dispositifs informatiques à connecter. Ni l'une ni l'autre de ces tâches ne convient à une solution par un système informatique. Les deux tâches requièrent des techniques bien établies qui peuvent être aisément réalisées sur base de cas par cas. La troisième tâche confirme que différents systèmes informatiques représentent

des informations sous des formes différentes. Il est bien connu que, même dans des systèmes livrés par le même fabricant, les structures de données décrivant des objets du monde réel sont différentes dans des systèmes différents, même lorsque les objets du monde réel en cours de description sont identiques. La principale source de complexité et de retard dans la mise en oeuvre des interconnexions de systèmes réside dans la production des programmes nécessaires pour réaliser les conversions requises des structures de données.

Avec cet arrière-plan général, on considérera à présent la mise en oeuvre de procédures de programmation à liaison retardée.

Les programmes de liaison retardée décrits sont constitués de deux parties - un programme de commande et des sous-programmes, qui effectuent les stades requis du programme. Le but du programme de commande est de recevoir les stimuli entrants, avec leurs données associées, et d'en tirer l'action appropriée à effectuer. L'action prend la forme d'une liste de sous-programmes susceptibles d'être appelés et de l'ordre dans lequel ces programmes doivent être appelés. Le programme de commande appelle les sous-programmes requis dans l'ordre requis, en vérifiant que chaque appel a été couronné de succès.

Dans la forme de réalisation préférée de la présente invention, les programmes peuvent être écrits dans un langage quelconque, de préférence dans le langage de programmation C. Le programme de commande extrait ses instructions de la base de données et les conserve sous la forme de réseaux de structures de données dans la mémoire de l'ordinateur. (Cette extraction de structures de données

dans des structures de mémorisation n'est pas essentielle, mais elle permet de réaliser le programme bien plus rapidement.) L'un des réseaux de données est constitué de toutes les séries d'appel de sous-programmes. Les structures de données contiennent une référence de fonction exécutable (en langage C, il s'agit d'un "indicateur de fonction") au sous-programme qui est la référence requise par le programme pour exécuter le sous-programme. Cet indicateur de fonction ne peut être maintenu dans la base de données. Pour déterminer les indicateurs de fonction de tous les sous-programmes requis, la base de données contient une chaîne unique de caractères pour chaque sous-programme disponible (c'est le nom de la fonction qui est en réalité codé dans chaque sous-programme). Le programme de commande contient une fonction qui peut convertir ces noms de fonction en indicateurs de fonction. Cette opération est réalisée par le programme de commande lorsqu'il commence tout d'abord l'exécution, ce qui enlève toute nécessité de le réaliser pour chaque stimulus du programme.

Aucun traitement d'application n'a lieu dans le programme de commande. Ses responsabilités concernent intégralement la sélection, le séquencement et l'exécution des procédures. Le programme de commande extrait les informations dont il a besoin d'une base de données, qui, pour les besoins de la description, peut être dénommé "métabase de données". Les changements de la métabase de données se refléteront dans le changement de comportement du programme. Le procédé est représenté dans la Fig. 7. La séquence est commandée par le programme de commande sous la direction de la métabase de données.

Sur base de ces principes, de nouveaux programmes peuvent être créés et modifiés sans changer le programme de commande. Ce concept peut être utilisé à la fois comme procédure de création de programmes ainsi que dans un système de traduction qui traduit une source d'informations en une autre. La raison en est que, une fois que le programme de commande a établi les stades de base, la traduction de données d'un format en données d'un autre format sont des opérations englobées dans des procédures et dans la structure de base de données avec et selon laquelle les procédures opèrent.

Le programme de traduction (Fig. 4) opère de la manière suivante : une fois que les données à traduire ont été obtenues (c'est-à-dire, des données de source), elles doivent être identifiées. Cela signifie que le programme de traduction doit reconnaître lequel des ensembles de messages entrants possibles ont été reçus. Cette identification détermine l'ensemble des structures de sortie (c'est-à-dire, des données de cible) qui doivent être acheminées au récepteur. Cet ensemble peut contenir zéro, une ou plusieurs structures de sortie. Cette traduction peut être décomposée en de nombreux petits stades de traduction, dont chacun s'applique à une valeur simple des données de source.

La décomposition est choisie de telle sorte que le type de données de chaque sous-section de la structure originale soit constitué d'un type unique de données lisibles par l'ordinateur (c'est-à-dire un nombre entier, un nombre à virgule flottante, une configuration binaire ou une chaîne de caractères). Par suite, la capacité de conversion libre entre des types de données peut résoudre

énormément des traductions de bas niveau requises. En examinant les transformations qui ne sont pas purement structurelles, on peut constater que la majorité d'entre elles ne sont pas sujettes à une solution algorithmique. Ce sont normalement des conversions de texte arbitraire d'une valeur à une autre. Un exemple en est la conversion d'un code d'une unité monétaire d'une forme arbitraire (par exemple, celle définie par Reuters) en une autre (par exemple, celle définie par Swift). Il n'y a aucun changement de la valeur sémantique des données; la source et la cible représentent toutes deux une monnaie, mais il n'y a pas de liaison algorithmique entre les valeurs. La solution pour ce type de transformation consiste à établir une table assortie de la plage des valeurs de source possibles et des valeurs de cible correspondantes, et un mécanisme généralisé pour rechercher la table et substituer la valeur de cible dans la structure de cible.

L'ensemble de sous-programmes nécessaires pour mettre en oeuvre un programme de traduction consiste donc en un ensemble de programmes de conversion structurels pour réaliser des fonctions telles que:

* une copie (avec justification à gauche ou à droite, selon ce qui s'avère approprié)

* une conversion d'un format de caractère en format numérique entier (et vice versa),

* une conversion de format de caractère en format numérique à virgule flottante (et vice versa),

* une conversion de format à virgule flottante en format numérique entier (et vice versa),

* une consultation de fichiers de données entrants dans le tableau de valeurs (la sortie se présente

sous la forme de caractères, de nombres entiers ou de virgules flottantes).

Une autre liste de fonctions représentatives est annexée dans l'appendice. Lorsqu'on l'appelle, chacun de ces programmes reçoit une référence aux données qu'il doit prendre comme entrée, une référence à l'emplacement où sa sortie doit être transcrite, les longueurs maxima autorisées de l'entrée et de la sortie, et une référence au tableau de consultation (le cas échéant) utilisé.

Le programme de traduction est donc à même de réaliser n'importe quelle traduction de données concevables dans le cadre des paramètres de structure de données qui peuvent être décrites par le langage de programmation C qui est à même de décrire n'importe quelle structure de données qui peut être représentée sur un ordinateur numérique moderne.

L'action d'appel d'un sous-programme exige que le programme d'appel ait trois informations sur le sous-programme :

* une référence au code de sous-programme (habituellement le nom de la fonction ou un indicateur de fonction)

* les données requises comme paramètres par le sous-programme

* les données à renvoyer à l'appelant par le sous-programme.

Pour que le programme de commande exprime un algorithme quelconque, la totalité de ces trois caractéristiques doit être constante ou indépendante du programme de commande d'appel. La référence de sous-programme est rendue indépendante via le mécanisme

indicateur de fonction décrit ci-dessus. Dans le programme de traduction, les données paramétriques sont constantes pour tous les sous-programmes de traduction concevables, comme décrit ci-dessus. Les données renvoyées par un sous-programme sont limitées à une valeur numérique, indiquant un succès ou une défaillance.

Ces restrictions - cet ensemble de paramètres fixes et une valeur de retour fixe - décrivent les exigences d'une sous-classe particulière de sous-programmes qui sont connus sous le nom de procédures de traduction. Ces sous-programmes sont à même d'être assemblés dans un programme de traduction concevable quelconque.

Les programmes de traduction utilisent un type différent de procédure, les procédures d'identification, pour effectuer l'identification de messages entrants. Ces procédures d'identification reçoivent à titre de paramètres l'emplacement du message reçu, le décalage et la longueur en octets de la zone du message entrant qu'elles doivent examiner et l'emplacement de la liste de messages de source, qui comprend des données arbitraires pour les identifier. Les procédures d'identification renvoient une valeur au programme de commande indiquant lequel, le cas échéant, des types de messages entrants a été reçu.

Les types d'algorithme qui peuvent être réalisés dans les procédures d'identification comprennent des comparaisons avec des valeurs de données connues de manière exacte, des comparaisons avec des valeurs de données partiellement connues (en utilisant des "expressions normales") ou une comparaison avec des types de données.

Les programmes de traduction utilisent des procédures de traduction pour toutes leurs activités de

traduction. Pour étendre le modèle de traitement du programme de commande et des sous-programmes afin d'inclure des programmes universels, il est nécessaire d'étendre la plage des sous-programmes que l'on peut exécuter. Il est particulièrement intéressant d'être à même d'utiliser des sous-programmes qui n'ont pas été construits à cet effet pour cet environnement. L'aptitude à réaliser un code bien établi conçu pour être mis en oeuvre dans d'autres programmes permet la construction extrêmement rapide d'un programme utilisant le programme de commande et une architecture de sous-programmes pour de nouvelles applications.

L'extension à une programmation universelle est obtenue en éliminant la restriction sur les données de paramètres utilisées par les sous-programmes. Un sous-programme qui est utilisé dans cet environnement est connu sous le nom de procédure d'application. Les procédures d'application sont des pièces fixes de code de programme que l'on peut obtenir de l'une des trois manières suivantes :

1. Ils peuvent être construits à la main.

2. Ils peuvent être construits en extrayant les fonctions primitives préexistantes de systèmes existants.

3. Ils peuvent être constitués de systèmes préexistants entiers.

Le programme de commande atteint sa généralité en traitant toutes les procédures de la même manière. Une procédure existe dans un environnement représenté dans la Fig. 5. Une procédure effectue un algorithme ou une partie d'un algorithme. Elle effectue cet algorithme sur des données. La procédure obtient des données à partir :

* d'une base de données à laquelle la procédure commande son propre accès,

* de données partagées avec d'autres procédures d'application (stockage global)

* de paramètres.

La totalité des trois types d'accès à des données sont effectués via des structures de données qui sont commandées entièrement dans la procédure d'application. Cela permet aux procédures d'être totalement indépendantes les unes des autres, sauf lorsque elles y sont spécifiquement autorisées, par inclusion de structures de données partagées. Les procédures sont toujours indépendantes du programme de commande. Le mécanisme par lequel la procédure d'application obtient des données d'une base de données ou via des structures de données partagées est fixé en tant que partie de sa structure interne. Ces mécanismes d'obtention de données sont complètement indépendants du programme de commande. Le programme de commande doit, cependant, fournir les données paramétriques à la procédure d'application de manière qu'elles puissent être librement réutilisées.

Pour permettre au programme de commande de le faire, un programme de traduction est utilisé. Lorsqu'un stimulus arrive au programme de commande, le programme de commande appelle son programme de traduction interne. Le programme de traduction crée une structure de données pour chaque procédure d'application à exécuter. La structure de données prend la forme représentée dans la fig 6. Cette structure est utilisée par une extension du programme de commande pour appeler la procédure désignée dans la référence de procédure, en faisant passer les paramètres

dans le restant de la structure.

Cette technique permet au programme de commande d'exécuter tout sous-programme qui est transcrit dans un langage informatique courant. Ainsi, du fait que le programme de commande est écrit en langage C, cela signifie que l'on peut utiliser n'importe quel langage qui peut être appelée forme C.

Il restera certains cas où le procédé d'appel du sous-programme est inapproprié ou qu'un autre langage ou une autre restriction de mise en oeuvre empêche un accès direct du programme de commande au sous-programme. Dans ce cas, un programme "emballeur" (ou une série d'emballeurs) est utilisé. Un "emballeur" est un sous-programme dont le but est déêtre appelable par le programme de commande et qui est à même d'appeler le programme de procédure d'application requis. Cette petite extension à la structure de procédures d'application permet à un concepteur de logiciels d'utiliser n'importe quel logiciel concevable en tant que procédure d'application.

Les stades de création du programme comprennent un stade individuel pour déterminer le besoin d'un programme de traduction et comprendre le cadre des tâches qu'il aura réalisées. Cela nécessite une détermination de la liste des messages entrants que l'ordinateur recevra. Une fois que la liste des messages possibles a été assemblée, on doit déterminer un mécanisme permettant de distinguer entre eux uniquement sur base de leur structure et de leur contenu. Le stade suivant est de déterminer pour chaque message entrant la réaction du traducteur. En d'autres termes, en réponse à un message ou un stimulus entrant, le traducteur ne peut créer aucun message de

sortie (c'est-à-dire, ignore le message entrant) ou crée un message ou plusieurs messages. Le nombre sera connu.

Le stade suivant consiste à décomposer les stades de programmation par identification des éléments de données simples qui constituent la structure de chaque message de cible. Cela se fait manuellement comme dans les stades précédents. Ensuite se fait l'identification des éléments les plus simples qui constituent la structure originale et qui correspondent aux éléments résultants dans le message de source. Pour chacun des éléments simples de la structure originale qui sont nécessaires à la source, on décide ensuite du type de transformation qui est nécessaire pour créer l'élément obtenu approprié. S'il y a des exigences quelconques de séquencement, ils doivent être pris en compte. Si, par exemple, il y a prise d'une moyenne, il est important que l'addition soit complétée avant la division. Dans tous les autres cas, un ordre arbitraire est acceptable. Le résultat de cette analyse est un ensemble d'instructions de transformation qui est constitué des éléments de données initiaux (c'est-à-dire, un champ dans la première structure), d'un élément de données résultant (c'est-à-dire, un champ dans la deuxième structure) et de la procédure de transformation requise entre eux. Chacun de ces éléments constitue une rangée de la table de détails XLAT de la métabase de données. Cela peut être mis à part dans la table de détails de traduction sous le nom de détails XLAT.

Les données sont ensuite entrées dans les tables de la métabase de données. L'existence d'un traducteur est enregistrée dans la table d'emplacements et on y enregistre également la procédure d'identification à appeler pour

distinguer entre les messages entrants et la section des messages entrants à examiner par la procédure d'identification. Chacun des types de messages entrants est enregistré dans la table de source conjointement avec des données arbitraires quelconques que l'on trouvera dans des messages de ce type de la section définie dans la table d'emplacements. Chacun des messages de cible de chaque message de source est enregistré dans la table de cibles. Chacun des détails de transformation est enregistré dans une table de détails de traduction.

Pour chaque transformation, on appelle une procédure (c'est-à-dire, un sous-programme qui effectue la transformation particulière requise) qui est représentée par une référence à la table des procédures dans la métabase de données. La table des procédures elle-même contient un nom pour un sous-programme exécutable (c'est-à-dire, une procédure). Ce nom est utilisé par le programme de commande pour en tirer la référence de fonction exécutable du sous-programme approprié lorsqu'il lit la base de données. Le résultat final pour chacune des transformations vient de cette référence. Il traduit ensuite la référence pour trouver le sous-programme et l'exécuter. Les autres informations qu'il doit passer au sous-programme et qui constituent des informations paramétriques sont constituées d'un indicateur pour indiquer où le champ de sources se trouve dans sa mémoire informatique temporaire, où le champ de cibles se trouve dans sa mémoire informatique temporaire, la taille de ces champs en octets et, si nécessaire, une référence à une table de consultation qui serait utilisée dans des transformations non algorithmiques. En d'autres termes, le

procédé d'analyse peuple la base de données en termes de formes et de structures de messages qui ont besoin d'être traduits et le programme de commande interprète ceux-ci comme instructions pour effectuer la traduction.

Le programme de commande existe dans un monde où il reçoit des stimuli. En fait, on lui fournit des données entrantes sur lesquelles il peut opérer.

Jusqu'à présent, le programme de commande est écrit pour extraire les données de la métabase de données et tirer les instructions de ces données pour savoir quelle procédure doit être appelée et avec quels stimuli. Il est impérieux qu'il n'y ait rien dans le programme de commande qui se réfère à une autre activité quelconque. Le programme de commande lit la métabase de données et utilise les instructions qu'il y trouve pour manipuler les structures de données qu'on lui présente.

Un procédure est celle qui effectue réellement la manipulation. Par exemple, l'une des instructions pourrait être "copier le troisième champ à l'entrée dans le sixième champ à la sortie". Le programme de commande localise le troisième champ en termes d'adresses d'octets. Le programme de commande consulte également la métabase de données pour connaître la taille du troisième champ. Le stade suivant effectué par le programme de commande consiste à localiser la sixième champ dans le message de cible, s'il se trouve, et à contrôler sa taille en octets. Le stade suivant effectué par le programme de commande, conformément aux directives reçues de la métabase de données, consiste à exécuter une procédure qui peut être, par exemple, "copier". Il exécute cette procédure en lui passant la position et la longueur de la cible et de la source. La

procédure effectue l'opération de copie de la position de source à la position de cible. Il est à noter que le programme de commande n'effectue pas la copie. Cependant, il appelle simplement quelque chose (c'est-à-dire, une procédure) dont la référence de fonction exécutable est dérivée du mot "copie" qu'il a retrouvé dans la métabase de données. Comme dans cet exemple, dans n'importe quel autre type de transformation, il se produit exactement le même processus. Le programme de commande, sous la direction des instructions de la métabase de données, calcule la taille et la position des données de source et des données de cible et appelle la procédure appropriée, en lui passant ces informations. Les activités du programme de commande sont limitées au calcul de paramètres et à l'appel de procédures. Il ne "sait" pas ce que les procédures font. En d'autres termes, c'est un programme neutre.

Chaque fois qu'une procédure s'exécute, la procédure exécutée avise le programme de commande qu'elle a réalisé ou non avec succès ce stade opérationnel. Le programme de commande examine cette valeur renvoyée pour assurer que chaque procédure a été couronnée de succès, en vérifiant de la sorte que la transformation source - cible a été réalisée avec succès. Lorsque le programme de commande a exécuté toutes les procédures qu'il est censé réaliser, il délivre la structure de sortie au récepteur. Le programme de commande attend alors le signal ou stimulus d'entrée suivant.

Dans une structure préférée de ce procédé, on admet que les types de messages qu'un traducteur donné reçoit changent très lentement en comparaison de la vitesse de réception de messages. En d'autres termes, un traducteur

peut s'attendre à recevoir de nombreuses copies d'une structure donnée de message avant que la structure de ce message ne soit modifiée par l'émetteur. Le processus par lequel passe l'ordinateur pour examiner la métabase de données conservée dans des fichiers sur disque prend très peu de temps. Il doit lire une grande quantité de fichiers de la base de données, ce qui, comme il est bien connu, est très complexe et difficile. L'ordinateur est toujours nécessaire pour effectuer les mêmes calculs chaque fois qu'un stimulus est reçu, à de nombreuses reprises : où se trouve le premier champ, où se trouve le quatrième champ ? Une voie préférée était de construire un programme de traduction pour tous les calculs de positions et de longueurs et la traduction de toutes les procédures de noms de texte en indicateurs de fonction est réalisée lorsque le programme de commande commence le traitement pour la première fois. Le programme de commande stocke alors une version abrégée de sa partie de la métabase de données dans une partie allouée de la mémoire RAM. Cela a pour effet de former les réseaux de données dont il a été question précédemment. Le seul objet de cette mémoire de stockage est d'accélérer le fonctionnement du programme. On a observé que cette mémoire de stockage améliorait la vitesse de fonctionnement d'un facteur d'environ 1.000. Le concept est similaire à l'emploi d'une antémémoire dans le processus de lecture et d'écriture dans des fichiers sur disque, si ce n'est que, dans l'antémémoire, il ne se fait aucune manipulation de données. Dans une antémémoire, un ordinateur écrit et copie des instructions répétées. Ici, dans ce cas, l'ordinateur non seulement lit dans la mémoire, ce que l'antémémoire fait également, mais il

convertit aussi les informations en une forme plus directement utilisable. La création de cette version de la métabase de données se produit lorsque le programme commence le traitement pour la première fois. Lorsque le programme de commande reçoit un message entrant, un stimulus, au lieu de se référer à la métabase de données proprement dite, il utilise les structures de la mémoire interne du programme de commande qui, comme indiqué, sont une version abrégée des mêmes données qui sont contenues dans la métabase de données.

Les procédures de traduction sont transcrites sous la forme de sous-programmes qui reçoivent un ensemble particulier de paramètres. Chaque procédure réalise un type particulier de transformation. Par exemple, il y en a une pour effectuer une copie et une pour effectuer la conversion de nombres de format de caractères en nombres entiers. Chacune des procédures réalise un type particulier de manipulation de types de données simples. Le nombre de types de données simples est très faible. En conséquence, on dispose d'un ensemble complet de ces procédures.

Les procédures d'application sont différentes des procédures de traduction. Comme mentionné ci-dessus, le traducteur est celui qui a cette exigence que toutes les procédures de traduction prennent exactement les mêmes paramètres. Cette exigence consiste à limiter le programme de commande à l'interprétation des stades effectués par les procédures et de conserver la neutralité du programme de commande. Cette neutralité est essentielle à la présente invention du fait qu'elle permet de modifier aisément la totalité du programme et de l'étendre sans effectuer de changements quelconques dans le programme de commande. Cela

permet l'addition de procédures pour que le programme entier puisse effectuer de nouvelles fonctions sans informer (c'est-à-dire modifier) le programme de commande.

Lorsqu'un sous-programme est exécuté dans un langage de programmation quelconque, une structure formelle est nécessaire. Une partie de cette structure formelle réside dans les paramètres qui sont envoyés aux sous-programmes. Le programme d'appel doit faire passer exactement les mêmes paramètres que les paramètres appelés que le programme s'attend à recevoir, autrement les résultats sont imprévisibles. En d'autres termes, le programme de commande qui exécute une procédure de traduction particulière et les paramètres des procédures de traduction sont très simples et seront toujours les mêmes. Dans le cas du traducteur, cela ne crée pas de problème, car toutes les procédures de traduction concevables nécessiteront les mêmes données paramétriques. On peut ainsi ajouter de nouvelles procédures de traduction pour effectuer de nouvelles tâches en toute impunité. Cela est dû au fait que les paramètres sont identiques.

Dans le cas de programmes universels, on ne peut jamais prédire ce que seront les paramètres nécessaires à une nouvelle procédure. Il est cependant important que le programme de commande soit à même d'accepter de nouvelles procédures avec leurs propres ensembles de paramètres nouveaux sans devoir modifier le programme de commande.

Ce problème est résolu en plaçant tous les paramètres nécessaires à une procédure d'application dans une structure de données. Cela oblige les procédures d'application à recevoir tous leurs paramètres sous la forme d'une structure. Dans le cas de procédures

d'application qui sont obtenues à l'aide d'un code transcrit pour d'autres systèmes, cela peut ne pas être correct. Cependant, c'est une manière simple pour un programme "emballeur" de convertir les données d'une structure en paramètres séparés. La structure paramétrique nécessaire à une procédure d'application est une structure de données qui peut être créée par un programme de traduction.

Pour résoudre ce problème, les procédures d'application ont différentes exigences. Les premières procédures d'application sont appelées différemment des procédures de traduction. Au lieu de donner aux procédures d'application leurs paramètres, ceux-ci sont stockés dans une table de la base de données. Toute information qu'un ordinateur peut conserver peut l'être dans une structure de données. Ainsi, dans un programme informatique ordinaire, un programme peut appeler un sous-programme ordinaire avec ses paramètres séparés. Dans ce cas, le programme d'application est appelé, il se réfère à la base de données pour trouver un paramètre qui est une structure de données qui contient tous les paramètres dont la procédure a besoin. Ce que la procédure exige importe peu. En d'autres termes, il n'y a pas de limitation à l'idée d'une structure par opposition à des parties de données séparées. Les procédures d'application ont chacune cette exigence que leurs paramètres se présentent sous la forme d'une structure stockée. La seule limite à la procédure d'application est la manière dont ses paramètres sont réglés.

En appliquant les techniques d'un traducteur à la procédure d'application, on construit un programme

d'application. Dans le premier stade, on décide si un programme d'application est nécessaire. Le programmeur fait la liste de tous les messages de stimuli que le programme d'application recevra. Comme dans le traducteur, chacun des stimuli va dans une table de sources. Une liste est établie de chaque opération, des procédures d'application qui effectueront l'opération requise et toutes les exigences de classement. Chacun de ces appels des procédures d'application est enregistré sous la forme d'une rangée dans la table de cibles. En d'autres termes, chacun de ces appels est à présent un message. Chacune de ces cibles est un message spécial qui est constitué de deux parties (se référer à la Fig. 6). La première partie du message est toujours une référence à une procédure d'application. La deuxième partie est une référence à la structure paramétrique dont a besoin la procédure. Le stade suivant est de créer le message de cible à partir du stimulus. Cela se fait de la même manière qu'avec le traducteur séparé en identifiant la partie du message entrant du message source qui crée chacun des paramètres et ensuite en utilisant l'une des procédures de l'invention qui est "juste réglée à une valeur". En d'autres termes, pour créer le nom de la fonction de la procédure au début du message, on utilise une procédure qui règle une valeur prise dans une table de consultation. (Cela est dû au fait que l'on ne s'attend pas à voir une partie quelconque du message du stimulus qui se réfère aux noms de fonction des procédures.). De la sorte, il résulte que le programme se comporte exactement comme un traducteur. Ensuite, au lieu de délivrer le message à un récepteur sous la forme d'un message de cible, ce que fait un traducteur, un nouvelle partie du programme de commande

(connue sous le nom d'interpréteur serveur) l'utilise de manière interne. Il traduit la référence de la procédure de la même manière que le traducteur traduit ses procédures de traduction. Il convertit donc la référence en un indicateur de fonction et appelle ainsi cette procédure en faisant passer une référence d'indicateur au restant du message sous la forme d'un paramètre. Le programme de commande contrôle alors les codes de retour comme dans le traducteur pour compléter le procédé, comme ci-dessus.

On a annexé à la présente demande sous la forme d'un appendice les formes de réalisations préférées de la documentation de conception du traducteur et des interpréteurs serveurs qui constituent le programme d'écriture d'application. Les diagrammes sont des organigrammes du type Gane et Sarson avec des descriptions qui leur sont attachées.

Le programme de commande est un programme qui appelle des stades de calcul de base sous la forme de procédures. Il exige des bases de données pour lui dire laquelle de ces procédures doit être utilisée et dans quelle circonstances d'entrée de données. Les stades de base sont créés à l'extérieur du programme de commande et de nouveaux stades peuvent toujours être ajoutés sans modifier le programme de commande. En conséquence, tout changement effectué au programme doit l'être dans les méthodes ou les bases de données que les procédures utilisent. Ce procédé élimine la nécessité de réécriture des programmes.

REVENDICATIONS

1. Procédure de programmation machine à liaison retardée pour agir sur des messages de source lors du réglage d'un ordinateur du type ayant une mémoire temporaire et dans lequel le programme traduit un message de source prédéterminé en un message de cible prédéterminé, le programme de liaison retardée comprenant les stades suivants :

(a) on utilise un programme de commande,

(b) on délivre les messages de source au programme de commande,

(c) on utilise une base de données,

(d) on stocke des instructions et des données dans la base de données,

(e) on amène le programme de commande à accéder à la base de données,

(f) on amène le programme de commande à répondre à des instructions et des données stockées dans la base de données,

(g) on utilise au moins un programme de procédure ou sous-programme de traduction,

(h) on amène le programme de commande, sensible à des instructions stockées dans la base de données, à appeler le programme de procédure de traduction,

(i) on amène le programme de procédure de traduction, en réponse à des instructions de la base de données, à traduire au moins une partie du message reçu, et

(j) on limite le programme de commande à répondre au message de source, à accéder à la base de données et à exécuter le programme de procédure de traduction en réponse

à des instructions et des données stockées dans la base de données.

2. Programme de liaison retardée selon la revendication 1, dans lequel le stade de stockage d'instructions et de données comprend le stockage des instruction d'appel de programme de procédure de traduction sur lesquels doit agir le programme de commande, et le stade de délivrance du programme de procédure de traduction comprend la délivrance d'un programme de procédure de traduction pour opérer sur au moins une partie du message de source.

3. Programme de liaison retardée selon la revendication 2, dans lequel le stade de délivrance d'instructions et de données à la base de données comprend la délivrance d'une chaîne unique de caractères ou de noms de fonctions pour chaque programme de procédure d'application et la délivrance au programme de commande d'une fonction pour convertir des noms de fonctions en références de fonctions exécutables.

4. Programme de liaison retardée selon la revendication 3, dans lequel le stade qui amène le programme de commande à répondre aux instructions et données de la base de données comprend le fait d'amener le programme de commande à répondre aux instructions et données au démarrage du programme de liaison retardée.

5. Programme de liaison retardée selon la revendication 4, dans lequel le stade qui est à même d'amener le programme de commande à répondre aux instructions de la base de données, au démarrage du programme de liaison retardée, amène le programme de commande à accéder à la base de données pour retirer de la

base de données les noms de fonctions.

6. Programme de liaison retardée selon la revendication 5, dans lequel le stade qui vise à amener le programme de commande à répondre aux instructions de la base de données amène le programme de commande à commander la séquence de stades de programmation sous la directive d'instructions et de données stockées dans la base de données.

7. Programme de liaison retardée selon la revendication 6, dans lequel le stade de stockage de données et instructions dans la base de données comprend la mise en oeuvre de tables assorties de la plage des valeurs possibles et d'un mécanisme généralisé pour rechercher la table susceptible d'être utilisée avec des traductions non algorithmiques.

8. Programme de liaison retardée selon la revendication 7, dans lequel le stade de délivrance d'un programme de procédure de traduction comprend la mise en oeuvre d'une série de programmes de procédures de traduction, le stade d'exécution d'un programme de procédure de traduction consiste à amener le programme de commande à accéder à la base de données pour obtenir une référence sur l'emplacement des données de source, une référence à l'emplacement des données de cible sur lesquelles les données de sources traduites doivent être transcrites, les longueurs maxima de chaque donnée de source et donnée de cible et une référence à une table de consultation quelconque établie dans les instructions de la base de données.

9. Programme de liaison retardée selon la revendication 8, dans lequel le stade de traduction

comprend les stades de copie (avec justification sur la gauche ou la droite, comme déterminé par les données de source et de cible), de conversion de format de caractères en format numérique entier (et vice versa), de conversion de format de caractères en format numérique à virgule flottante (et vice versa), de conversion du format à virgule flottante en format numérique entier (et vice versa), de consultation de fichiers de données entrantes dans un tableau de valeurs (sortie sous la forme de caractères, de nombres entiers ou de virgules flottantes).

10. Programme de liaison retardée selon la revendication 8, dans lequel le stade d'appel du programme de procédure de traduction comprend la délivrance par la base de données d'instructions et de données préalablement stockées comprenant la délivrance d'une référence au code de programme de procédure de traduction, des données requises sous la forme de paramètres par le programme de procédure de traduction et des données à renvoyer à l'instruction d'appel provenant de la base de données par le programme de procédure de traduction.

11. Programme de liaison retardée selon la revendication 10, dans lequel le stade de délivrance d'une référence au code de programme de procédure de traduction comprend la délivrance d'un nom de fonction stocké dans la base de données.

12. Programme de liaison retardée selon la revendication 11, dans lequel le stade visant à amener le programme de commande à répondre aux instructions et données comprend par ailleurs la déduction par le programme de commande des indicateurs de fonction à partir des noms de fonctions stockés dans la base de données.

13. Programme de liaison retardée selon la revendication 12, dans lequel le stade visant à amener le programme de méthode de traduction à traduire comprend l'accès du programme de procédure de traduction à la base de données.

14. Programme de liaison retardée selon la revendication 10, dans lequel un programme de procédure de traduction renvoie les données au programme de commande en indiquant le succès ou la défaillance de traduction des données de source en données de cible.

15. Programme de liaison retardée selon la revendication 13, comprenant la nécessité de fixer les paramètres des procédures de traduction.

16. Programme de liaison retardée selon la revendication 13, comprenant la nécessité de prédéterminer la structure des messages de cible et de source.

17. Programme de liaison retardée selon la revendication 9, dans lequel le stade de production comprend par ailleurs la copie, octet par octet, avec une justification à droite ou à gauche; la consultation de la valeur de source dans une table assortie et le réglage de la cible à la valeur correspondant à la valeur de source; la consultation de la valeur de source dans une table assortie et le réglage de la cible à la valeur correspondant à la valeur de source, convertie en un nombre entier court; la consultation de la valeur de source dans une table assortie et le réglage de la cible à la valeur correspondant à la valeur de source convertie en nombre entier; la consultation de la valeur de source dans une table assortie et le réglage de la cible à la valeur correspondant à la valeur de source convertie en un nombre

entier long; la consultation de la valeur de source dans une table assortie et le réglage de la cible à la valeur correspondant à la valeur de source, convertie en un nombre à virgule flottante, la consultation de la valeur de source dans une table assortie et le réglage de la cible à la valeur correspondant à la valeur de source convertie en un nombre à virgule flottante à double précision; la conversion de caractères (ne contenant que des caractères numériques, des séparateurs facultatifs pour les milliers et les espaces) en la valeur du nombre entier court correspondante; la conversion de caractères (ne contenant que des caractères numériques, des séparateurs facultatifs pour les milliers et des espaces) en valeur de nombre entier correspondante; la conversion de caractères (ne contenant que des caractères numériques, des séparateurs facultatifs pour les milliers et des espaces) en valeur de nombre entier long correspondante; la conversion de caractères (ne contenant que des caractères numériques, des séparateurs facultatifs pour les milliers, un caractère décimal facultatif et des espaces) en la valeur à virgule flottante correspondante, la conversion de caractères (ne contenant que des caractères numériques, des séparateurs facultatifs pour les milliers et un caractère décimal facultatif et des espaces) en valeur à virgule flottante à double précision; la conversion d'un nombre entier court en un champ de caractères, la conversion d'un nombre entier en un champ de caractères; la conversion d'un nombre entier long en un champ de caractères; la conversion d'un nombre à virgule flottante en un champ de caractères; la conversion d'un nombre à virgule flottante à double précision en un champ de caractères; la conversion d'un

nombre à virgule flottante à double précision en un champ d'IMAGES, comme défini dans le langage de programmation COBOL; la conversion d'un champ d'IMAGES, comme défini dans le langage de programmation COBOL, en un nombre à virgule flottante à double précision; le réglage d'une valeur arbitraire dans le champ de cibles; l'addition de la valeur de source au champ de cibles à nombre entier court; l'addition de la valeur de source au champ de cibles à nombre entier; l'addition de la valeur de source au champ de cibles à nombre entier long; l'addition de la valeur de source au champ de cibles à virgule flottante; l'addition de la valeur de source au champ de cibles à virgule flottante à double précision; la soustraction de la valeur de source du champ de cibles à nombre entier court; la soustraction de la valeur de source du champ de cibles à nombre entier; la soustraction de la valeur de source du champ de cibles à nombre entier long; la soustraction de la valeur de source du champ de cibles à virgule flottante; la soustraction de la valeur de source du champ de cibles à virgule flottante à double précision; la multiplication de la valeur de source par le champ de cibles à nombre entier court; la multiplication de la valeur de source par le champ de cibles à nombre entier; la multiplication de la valeur de source par le champ de cibles à nombre entier long; la multiplication de la valeur de source par le champ de cibles à virgule flottante; la multiplication de la valeur de source par le champ de cible à virgule flottante à double précision, la division de la valeur de source par le champ de cibles à nombre entier court, la division de la valeur de source par le champ de cibles à nombre entier, la division de la valeur de source par le champ de cibles à

nombre entier long, la division de la valeur de source par le champ de cibles à virgule flottante, la division de la valeur de source par le champ de cibles à virgule flottante à double précision, l'application d'une opération logique binaire ET entre le champ de sources et le champ de cibles, l'application d'une opération logique binaire INCLUSIF OU entre le champ de sources et le champ de cibles, l'application d'une opération logique binaire EXCLUSIF OU entre le champ de sources et le champ de cibles.

18. Procédure de programmation d'ordinateur à liaison retardée pour créer des programmes, dans laquelle il est délivré au programme d'ordinateur à liaison retardée des stimuli de source dans lesquels chaque stimulus est du type comprenant un événement d'un ensemble d'un nombre prédéterminé d'événements et une structure d'un ensemble prédéterminé de structures de données, et dans laquelle le programme machine à liaison retardée est du type qui peut appeler au moins un sous- programme (c'est-à-dire, un "programme de procédure d'application"), la procédure du programme machine à liaison retardée comprenant les stades suivants :

(a) on utilise un programme de commande,

(b) on délivre les stimuli de source au programme de commande,

(c) on utilise une base de données,

(d) on stocke des instructions, des données et les structures de stimuli de source quelconques sur lesquelles on doit intervenir dans la base de données,

(e) on amène le programme de commande à accéder à la base de données,

(f) on amène le programme de commande à répondre

à des instructions et des données stockées dans la base de données,

(g) on amène le programme de commande, en réponse aux stimuli et à des instructions et des données de la base de données, à être à même d'appeler au moins un programme du procédure d'application, et

(h) on limite le programme de commande à répondre au stimulus de source, à accéder à la base de données et à répondre à des instructions et des données stockées dans la base de données.

19. Programme de liaison retardée selon la revendication 18, comprenant par ailleurs la mise en oeuvre d'un ensemble de programmes de procédures d'application prédéterminés.

20. Programme de liaison retardée selon la revendication 19, comprenant par ailleurs le stockage dans la base de données des paramètres de tous les programmes de procédures d'application prédéterminés, et dans lequel le stade qui amène le programme de commande à répondre au stimulus de source comprend la transmission par le programme de commande, sous la directive d'instructions et de données retrouvées par le programme de commande dans la base de données, de structures prédéterminées de paramètres à des programmes de procédures d'application prédéterminés.

21. Programme de liaison retardée selon la revendication 20, dans lequel le stade de stockage comprend le stockage des structures paramétriques dans la base de données.

22. Programme de liaison retardée selon la revendication 20, dans lequel le stade de transmission de paramètres par le programme de commande comprend par

ailleurs les instructions et les données retrouvées par le programme de commande et déterminant lequel des programmes de procédures d'application prédéterminés quelconques doit être appelé.

23. Programme de liaison retardée selon la revendication 20, dans lequel le stade d'appel comprend la délivrance de paramètres aux programmes de procédures d'application.

24. Programme de liaison retardée selon la revendication 20, comprenant par ailleurs la mise en oeuvre d'emballeurs, la délivrance au programme de commande de la capacité d'appeler au moins un emballeur, l'emploi des emballeurs pour appeler des programmes de procédures d'application ou d'autres emballeurs.

25. Programme de liaison retardée selon la revendication 20, dans lequel le stade de stockage d'instructions et de données dans la base de données comprend le stockage de l'appel du programme de procédure d'application, et le programme de commande, en réponse au stimulus et aux informations obtenues dans la base de données, étant à même d'appeler au moins un programme de procédure d'application prédéterminé.

26. Programme de liaison retardée selon la revendication 24, dans lequel le stade de délivrance d'instructions et de données à la base de données comprend la délivrance d'une chaîne unique de caractères ou de noms de fonctions pour chaque programme de procédure d'application ou emballeur, la délivrance au programme de commande d'une fonction permettant de convertir les noms de fonctions en indicateurs de fonctions.

27. Programme de liaison retardée selon la

revendication 26, comprenant par ailleurs des moyens pour traduire des données prédéterminées des stimuli en structures de paramètres prédéterminées du programme de procédure d'application et en emballeurs.

28. Programme de liaison retardée selon la revendication 25, qui comprend par ailleurs l'emploi du programme de commande pour commander la séquence d'appel des programmes de procédures d'application sous la directive d'instructions et de données stockées dans la base de données.

29. Programme de liaison retardée selon la revendication 27, dans lequel le stade de stockage des données et instructions dans la base de données comprend la mise en oeuvre de tables assorties de la plage de valeurs possibles et un mécanisme généralisé pour y chercher la table dans le cas de traductions non algorithmiques.

30. Programme de liaison retardée selon la revendication 25, dans lequel le stade de délivrance d'un programme de procédure d'application comprend la mise en oeuvre d'une série de programmes de procédures d'application, le stade d'exécution d'un programme de procédure d'application comprend le fait d'amener le programme de commande à accéder à la base de données pour obtenir la structure paramétrique pour le programme de procédure d'application, une référence à l'emplacement des données de stimuli, une référence à l'emplacement des paramètres des procédures d'application qui doivent y être transcrits, les longueurs maxima de chaque élément de la structure de stimulus, la longueur maximum des données de stimulus et de la structure paramétrique, et une référence à une table de consultation quelconque établie dans les

instructions de la base de données.

31. Programme de liaison retardée selon la revendication 28, dans lequel le stade de traduction comprend les stades de copie (avec justification à gauche ou à droite comme déterminé par les données de stimulus et de la structure paramétrique de la procédure d'application), de conversion de format de caractères à un format numérique entier (et vice versa), de conversion de format de caractères à un format numérique à virgule flottante (et vice versa), de conversion de format à virgule flottante en format numérique entier (et vice versa), de consultation des données entrantes dans les tableaux de valeurs (sortie sous la forme de caractères, de nombres entiers ou de virgules flottantes).

32. Programme de liaison retardée selon la revendication 26, dans lequel le stade d'appel d'un programme de procédure d'application ou emballeur comprend la délivrance par la base de données d'instructions et de données stockées comprenant la mise en oeuvre d'une référence au code de programme de procédure d'application ou code d'emballeur, des données requises en tant que paramètres par le programme de procédure d'application ou emballeur, et des données à renvoyer au programme de commande par le programme de procédure d'application ou emballeur.

33. Programme de liaison retardée selon la revendication 26, dans lequel le stade de délivrance d'une référence au programme de procédure d'application comprend la délivrance de noms de fonctions stockés dans la base de données.

34. Programme de liaison retardée selon la

revendication 26, dans lequel chaque programme de procédure d'application fournit des indices au programme de commande et dans lequel les indices indiquent le succès ou la défaillance de l'opération de ce programme de procédure d'application.

35. Programme de liaison retardée selon la revendication 34, dans lequel le stade de délivrance d'indices comprend, lorsque le programme de commande appelle le programme de procédure d'application, le passage des indices directement au programme de commande à partir du programme de procédure d'application.

36. Programme de liaison retardée selon la revendication 34, dans lequel le stade de délivrance d'indices comprend, au moment où au moins un emballeur appelle le programme de procédure d'application, le passage des indices via l'emballeur au programme de commande à partir du programme de procédure d'application.

37. Programme de liaison retardée selon la revendication 31, dans lequel le stade de production comprend par ailleurs la copie, octet par octet, avec une justification à droite ou à gauche; la consultation de la valeur de source dans une table assortie et le réglage de la cible à la valeur correspondant à la valeur de source; la consultation de la valeur de source dans une table assortie et le réglage de la cible à la valeur correspondant à la valeur de source, convertie en un nombre entier court; la consultation de la valeur de source dans une table assortie et le réglage de la cible à la valeur correspondant à la valeur de source convertie en nombre entier; la consultation de la valeur de source dans une table assortie et le réglage de la cible à la valeur

correspondant à la valeur de source convertie en un nombre entier long; la consultation de la valeur de source dans une table assortie et le réglage de la cible à la valeur correspondant à la valeur de source, convertie en un nombre à virgule flottante, la consultation de la valeur de source dans une table assortie et le réglage de la cible à la valeur correspondant à la valeur de source convertie en un nombre à virgule flottante à double précision; la conversion de caractères (ne contenant que des caractères numériques, des séparateurs facultatifs pour les milliers et les espaces) en la valeur du nombre entier court correspondante; la conversion de caractères (ne contenant que des caractères numériques, des séparateurs facultatifs pour les milliers et des espaces) en valeur de nombre entier correspondante; la conversion de caractères (ne contenant que des caractères numériques, des séparateurs facultatifs pour les milliers et des espaces) en valeur de nombre entier long correspondante; la conversion de caractères (ne contenant que des caractères numériques, des séparateurs facultatifs pour les milliers, un caractère décimal facultatif et des espaces) en la valeur à virgule flottante correspondante, la conversion de caractères (ne contenant que des caractères numériques, des séparateurs facultatifs pour les milliers et un caractère décimal facultatif et des espaces) en valeur à virgule flottante à double précision; la conversion d'un nombre entier court en un champ de caractères, la conversion d'un nombre entier en un champ de caractères; la conversion d'un nombre entier long en un champ de caractères; la conversion d'un nombre à virgule flottante en un champ de caractères; la conversion d'un nombre à virgule flottante à double

précision en un champ de caractères; la conversion d'un nombre à virgule flottante à double précision en un champ d'IMAGES, comme défini dans le langage de programmation COBOL; la conversion d'un champ d'IMAGES, comme défini dans le langage de programmation COBOL, en un nombre à virgule flottante à double précision; le réglage d'une valeur arbitraire dans le champ de cibles; l'addition de la valeur de source au champ de cibles à nombre entier court; l'addition de la valeur de source au champ de cibles à nombre entier; l'addition de la valeur de source au champ de cibles à nombre entier long; l'addition de la valeur de source au champ de cibles à virgule flottante; l'addition de la valeur de source au champ de cibles à virgule flottante à double précision; la soustraction de la valeur de source du champ de cibles à nombre entier court; la soustraction de la valeur de source du champ de cibles à nombre entier; la soustraction de la valeur de source du champ de cibles à nombre entier long; la soustraction de la valeur de source du champ de cibles à virgule flottante; la soustraction de la valeur de source du champ de cibles à virgule flottante à double précision; la multiplication de la valeur de source par le champ de cibles à nombre entier court; la multiplication de la valeur de source par le champ de cibles à nombre entier; la multiplication de la valeur de source par le champ de cibles à nombre entier long; la multiplication de la valeur de source par le champ de cibles à virgule flottante; la multiplication de la valeur de source par le champ de cible à virgule flottante à double précision, la division de la valeur de source par le champ de cibles à nombre entier court, la division de la valeur de source par le champ de cibles à nombre entier, la

division de la valeur de source par le champ de cibles à nombre entier long, la division de la valeur de source par le champ de cibles à virgule flottante, la division de la valeur de source par le champ de cibles à virgule flottante à double précision, l'application d'une opération logique binaire ET entre le champ de sources et le champ de cibles, l'application d'une opération logique binaire INCLUSIF OU entre le champ de sources et le champ de cibles, l'application d'une opération logique binaire EXCLUSIF OU entre le champ de sources et le champ de cibles.

38. Procédure d'écriture d'un programme de traduction pour traduire un message de source en un message de cible comprenant les stades suivants:

(a) on identifie les caractéristiques des paramètres du message de cible,

(b) on identifie les caractéristiques du message de source,

(c) on en déduit les conditions pour traduire le message de source en message de cible,

(d) on met en oeuvre une base de données,

(e) on stocke dans la base de données des données et instructions correspondant aux stades nécessaires pour traduire le message de source en message de cible,

(f) on écrit un programme de procédure de traduction pour chaque transformation de chaque type d'éléments du message de source en chaque type d'élément du message de cible,

(g) on stocke dans la base de données une référence au programme de procédure de traduction,

(h) on stocke dans la base de données des instructions identifiant les programmes de procédures de

traduction qui doivent être appelés,

(i) on limite le programme de commande de manière qu'il soit commandé à tout moment par les instructions et les données de la base de données, et

(j) on amène le programme de commande à obtenir dans la base de données, à la réception d'un message de source, les instructions de la base de données pour diriger le programme de commande et appeler des sous-programmes de procédures de traduction prédéterminés pour traduire le message de source en message de cible.

39. Procédure selon la revendication 38, consistant par ailleurs à faire en sorte que le programme de commande délivre le message de source traduit au récepteur.

40. Procédure selon la revendication 39, dans laquelle le stade d'identification des caractéristiques et des paramètres du message de cible et du message de source comprend l'identification des éléments de données les plus simples du message de cible et du message de source.

41. Procédure selon la revendication 40, dans lequel le stade de déduction de la traduction comprend la détermination du résultat de chaque traduction de chaque élément simple du message de source au message de cible.

42. Procédure selon la revendication 41, dans laquelle le stade de déduction de la traduction comprend par ailleurs la détermination du stade de traduction à entreprendre par le programme de traduction.

43. Procédure selon la revendication 42, dans laquelle le stade de déduction de la traduction comprend par ailleurs la détermination de la séquence des stades de traduction.

44. Procédure selon la revendication 43, dans laquelle le stade de stockage comprend le stockage des stades de traduction, l'entrée de données et d'instructions dans les tables, l'entrée des emplacements du programme de procédure de traduction et de la séquence d'appel du programme de procédure de traduction.

45. Procédure selon la revendication 44, consistant par ailleurs à faire en sorte que le programme de commande, au démarrage, crée une zone dans la mémoire temporaire de l'ordinateur, à stocker une table contenant des données utilisées pour identifier différents types de messages de source, à stocker des indicateurs dans la zone de mémoire temporaire pour identifier l'emplacement des champs de source dans le message de source, à stocker des tables de consultation dans la zone de mémoire temporaire pour des transformations non algorithmiques, à stocker des instructions dans la zone de mémoire temporaire pour identifier le programme de procédure de traduction qui doit être appelé et à stocker dans la zone de mémoire temporaire la séquence d'opérations qui permettra de diriger le programme de commande.

46. Procédure d'écriture d'un programme d'application comprenant les stades suivants:

(a) on fait la liste de tous les messages de stimuli que le programme d'application doit recevoir,

(b) on établit une liste de chaque opération que chaque programme de procédure de traduction d'application doit effectuer et la séquence des opérations,

(b) on établit une base de données,

(c) on stocke des stimuli dans une table dans la base de données,

(d) on stocke l'instruction pour exécuter le programme de procédure d'application dans la base de données sous la forme d'un message comprenant une référence au programme de procédure d'application et une référence aux paramètres que le programme de procédure d'application utilisera,

(e) on détermine chaque stimulus à recevoir par le programme d'application et on désigne les stimuli comme messages de source,

(f) on détermine les structures paramétriques pour chacun des programmes de procédures d'application à exécuter et on les désigne chacun comme messages de cible,

(g) on identifie les caractéristiques des paramètres de chaque message de cible,

(h) on identifie les caractéristiques du message de source,

(i) on en déduit les confitions pour traduire le message de source en messages de cible,

(j) on stocke dans la base de données les données et instructions correspondant aux stades nécessaires pour traduire le message de source en message de cible,

(k) on obtient un programme de procédure d'application pour chaque stade d'application,

(l) on stocke des instructions dans la base de données identifiant les programmes de procédures d'application qui doivent être appelés,

(m) on amène les instructions de la base de données à faire que le programme de commande appelle les programmes de procédures d'application et crée les paramètres associés à chaque programme de procédure d'application à partir d'éléments du stimulus de source,

(n) on limite le programme de commande de manière qu'il soit commandé à tout instant par les instructions et données de la base de données, et

(o) on amène le programme de commande à appeler la base de données à la réception d'un message de source, les instructions de la base de données à diriger le programme de commande et à appeler les programmes de procédures d'application prédéterminés pour traduire le message de source en un message de cible et délivrer le message de cible sous la forme d'au moins une partie du programme complet.

47. Procédure selon la revendication 46, dans laquelle le stade d'obtention consiste à écrire un programme de procédure d'application.

48. Procédure selon la revendication 46, dont le stade d'obtention comprend la copie d'un programme de procédure d'application.

A N N E X E .

Sélecteur de procédures d'application 02/05/06

Ce procédé est fonction de la version. Sa fonction est de recevoir des demandes partielles, de déterminer la procédure d'application requise et de l'exécuter, en faisant passer le restant de la demande partielle sous forme de paramètres. C'est ainsi qu'en CICS, le sélecteur de procédures d'application exécutera une liaison CICS et fera passer le restant des paramètres dans un COMMAREA. La procédure suivante décrit les actions du procédé.

Début
- **recevoir la demande partielle interne**
- **exécuter la Procédure d'Application identifiée par "application_method_function", en passant "Msg_Elements" comme paramètre**
- **recevoir "Response_Code" et "Msg_Elements" de la Procédure d'Application**

aller à Début

Procédures d'application 02/05/06

Les procédures d'application sont les unités atomiques du traitement d'application. Elles sont définies dans d'autres encyclopédies.

Toutes les procédures d'application reçoivent des paramètres que le métasystème considère comme étant un message consitué de "Msg_Elements" et renvoient un "Response_Code" et, éventuellement, des données par suite de leurs activités. Les données sont considérées par le métasystème comme un deuxième message constitué de "Msg_Elements". Les procédures d'application elles-mêmes manipulent des "Application_Data".

Interpréteur d'événements 02/05/06

L'interpréteur d'événements est le procédé de commande du serveur de l'application. Il peut être décrit par le pseudocode suivant :

- passer le message entrant (constitué de "Msg_Elements") au Traducteur Universel
- le premier message cible de traduction contient le nombre des réponses qui suivent
- pour chaque élément suivant du message cible de traduction et tant que l'on a pas d'erreur dans "Response_Code"
- passer "application_method_function" et "Msg_Elements" au sélecteur de procédures d'application
- recevoir "Response_Code"
fin pour
- si erreur
envoyer message d'erreur
sinon
envoyer message d'opération réussie
fin si

Gestionnaire de files d'attente 02/05/06

Le gestionnaire de files d'attente est un procédé qui est fonction de la version. Sa fonction est de recevoir des messages des files d'attente et de les placer sur des files d'attente.

Les messages entrants sont envoyés à un interpréteur d'événements pour traitement, les messages sortants sont reçus de l'interpréteur d'événements et envoyés à la file d'attente.

Les messages entrants sont visualisés comme "Msg_Elements", les message sortants sont des "Msg_Elements".

Traducteur en temps réel 10/26/92

"Ce processus effectue une traduction en temps réel d'un message entrant en zéro à plusieurs messages sortants. Le processus est contrôlé par translation_control_structure".

Début
- **recevoir le message source**
- **exécuter "Method_id" pour identifier le type du message**
- **pour "Target_First" à "Target_First" + "Target_Number" dans TARGET MESSAGE ARRAY**
  - construire le message cible champ par champ :
    pour "Field-First" à "Field_First" + "Field_Number"
      - **construire "Msg_Elements" à partir d'une section du message source prise à partir de l'adresse "FieldSrc_Offset" sur une longueur de "FieldSrc_Length" octets**
      - **construire "Msg_Elements" à partir d'une section du message cible prise à partir de l'adresse "FieldTrgt_Offset" sur une longueur de "FieldTrgt_Length" octets**
      - **appeler "Xlat_Method" en passant comme arguments le champ source, le champ cible et "LockupTable_Index"**

    fin pour

  fin pour

aller à Début

Extraits de définitions du traducteur 10/26/92

- **recevoir "Location_id"**
- **trouver le rang correspondant à "Location_id" dans la table de location du traducteur**
- **si pas trouvé**
  - **retourner un message d'échec**

  sinon
  - **assigner à "Source_Number" le nombre de rangs dans la table de traduction des sources pour lesquels "Location_id" correspond au "Location_id" reçu**
  - **assigner à "Method_id", "Src_id_Offset" et "Src_id_Offset" les valeurs correspondantes de la table de traduction**
  - **trouver le premier rang de la table de traduction pour lequel la correspondance de "Location_id" est établie**
  - **tant qu'on a correspondance**

    remplir le tableau de messages source
    - **assigner à "Src_id_Value" la valeur des sources de traduction**
    - **assigner à "Traget_First" l'indice disponible suivant dans le tableau des messages cible**
    - **assigner à "Target_Number" le nombre de rangs dans la table de traduction cible pour lesquels on a une correspondance "Location_id" et "Source_id"**
    - **trouver le premier rang de la table de traduction cible pour lequel on a correspondance "Location_id" et "Source_id"**
    - **tant qu'on a correspondance**

      construire une image temporaire du message source
      - **allouer de l'espace pour le tableau temporaire décrivant le message source; chaque élément contenant 2 entiers, qui sont l'adresse et la longueur du champ représenté, le nombre**

d'éléments étant égal au nombre de rangs dans les champs dans la table des messages pour lesquels "Msg_id" = "Source_id"

- pour chaque rang pour lequel on a correspondance "Msg_id" et "Source_id"
  - lire les champs de la table pour le rang correspondant à "Field_id"
  - sauvegarder "Field_Length" dans le tableau temporaire
  - calculer l'adresse = adresse de l'ancien champ + longueur de l'ancien champ + ajustement pour "Field_Align" dans la table des champs

fin pour

remplir le tableau des messages cible

- assigner à "Field_first" le premier indice disponible dans la table de traduction
- assigner à "Field_Number" le nombre de rangs dans la table des détails de traduction pour lesquels on a correspondance "Location_id", "Source_id" et "Traget_id"

construire une image temporaire du message cible

- allouer de l'espace pour le tableau temporaire décrivant le message cible; chaque élément contenant 2 entiers, qui sont l'adresse et la longueur du champ représenté, le nombre d'éléments étant égal au nombre de rangs dans les champs dans la table des messages pour lesquels "Msg_id" = "Target_id"
- pour chaque rang pour lequel on a correspondance "Msg_id" et "Target_id"
  - lire les champs de la table pour le rang correspondant à "Field_id"
  - sauvegarder "Field_Length" dans le tableau temporaire
  - calculer l'adresse = adresse de l'ancien

champ + longueur de l'ancien champ + ajustement pour "Field_Align" dans la table des champs

fin pour

remplir la table des champs de traduction

- trouver le premier rang de la table des détails de traduction pour lequel on a correspondance "Location_id, "Source_id" et "Target_id"
- tant qu'on a correspondance
  - lire la table des méthodes de traduction en recherchant un rang pour lequel on a correspondance "Method_id"
  - assigner à "Xlat_Method" "Method_Function" de la table des méthodes de traduction
  - obtenir "FieldSrc_Offset" et "FieldSrc_Length" de l'élément de la table temporaire de description du message source pour lequel l'indice vaut "FieldSrc_Seq"
  - obtenir "FieldTrgt_Offset" et "FieldTrgt_Length" de l'élément de la table temporaire de description du message cible pour lequel l'indice vaut "FieldTrgt_Seq"
  - si une table lookup est requise
    - si la table lookup est chargée
      assigner à "LookupTable_Index" l'indice de la table de lookup dans le tableau des tables lookup
    sinon
      remplir le tableau des tables lookup
      - assigner à "LkupTable_id" une valeur à partir de la table lookup
      - assigner à "LkupElem_Length" la somme "LkupSrc_Length" + "LkupTrgt_Length"
      - lire le rang de la table lookup correspondant à "LkupTable_id"
      - donner à la table lookup une longueur

```
                        égale à la somme "LkupSrc_Length" +
                        "LkupTrgt_Length"
                      - assigner à "LookupTable_Number" le
                        nombre de rangs dans la table lookup
                        correspondant à "LkupTable_id"
                      - assigner à "LookupTable_Offset"
                        l'adresse du premier octet disponible
                        dans "LkupData"
                      - charger tous les rangs de la table de
                        données lookup avec les
                        "LkupTable_id" correspondant aux
                        "LkupTable_id" de la table des
                        détails de traduction
                    fin si
                  fin si
                - trouver le rang suivant de la table des
                  détails de traduction pour lequel il y a
                  correspondance "Location_id", "Source_id" et
                  "Target_id"
              fin tant que
            - trouver le rang suivant de la table de
              traduction cible pour lequel on a
              correspondance "Location_id" et "Source_id"
          fin tant que
      fin si

  copier octet-par-octet
  copier le texte justifié à droite
  rechercher dans la table des littéraux
  rechercher un entier court
  rechercher un entier
  rechercher un entier long
  rechercher un nombre en virgule flottante
  rechercher un nombre en virgule flottante de double
  précision
  convertir d'ASCII numérique en virgule flottante double
```

précision
convertir d'ASCII numérique en virgule flottante
convertir d'ASCII numérique en entier court
convertir d'ASCII numérique en entier long
convertir d'ASCII numérique en entier
convertir d'entier en ASCII numérique
convertir d'entier court en ASCII numérique
convertir d'entier long en ASCII numérique
convertir de virgule flottante en ASCII numérique
convertir de virguke flottant double précision en ASCII numérique
convertir de virgule flottante double précision en Picture COBOL
convertir de Picture COBOL en virgule flottante double précision
assigner une valeur à une variable
ajouter à la cible (entier court)
ajouter à la cible (entier)
ajouter à la cible (entier long)
ajouter à la cible (virgule flottante)
ajouter à la cible (virgule flottante double précision)
soustraire de la cible (entier court)
soustraire de la cible (entier)
soustraire de la cible (entier long)
soustraire de la cible (virgule flottante)
soustraire de la cible (virgule flottante double précision)
multiplier par la cible (entier court)
multiplier par la cible (entier)
multiplier par la cible (entier long)
multiplier par la cible (virgule flottante)
multiplier par la cible (virgule flottante double précision)
diviser par la cible (entier court)
diviser par la cible (entier)
diviser par la cible (entier long)

diviser par la cible (virgule flottante)

diviser par la cible (virgule flottante double précision)

ET logique bit par bit

OU logique bit par bit

OU EXCLUSIF logique bit par bit

FIG.1

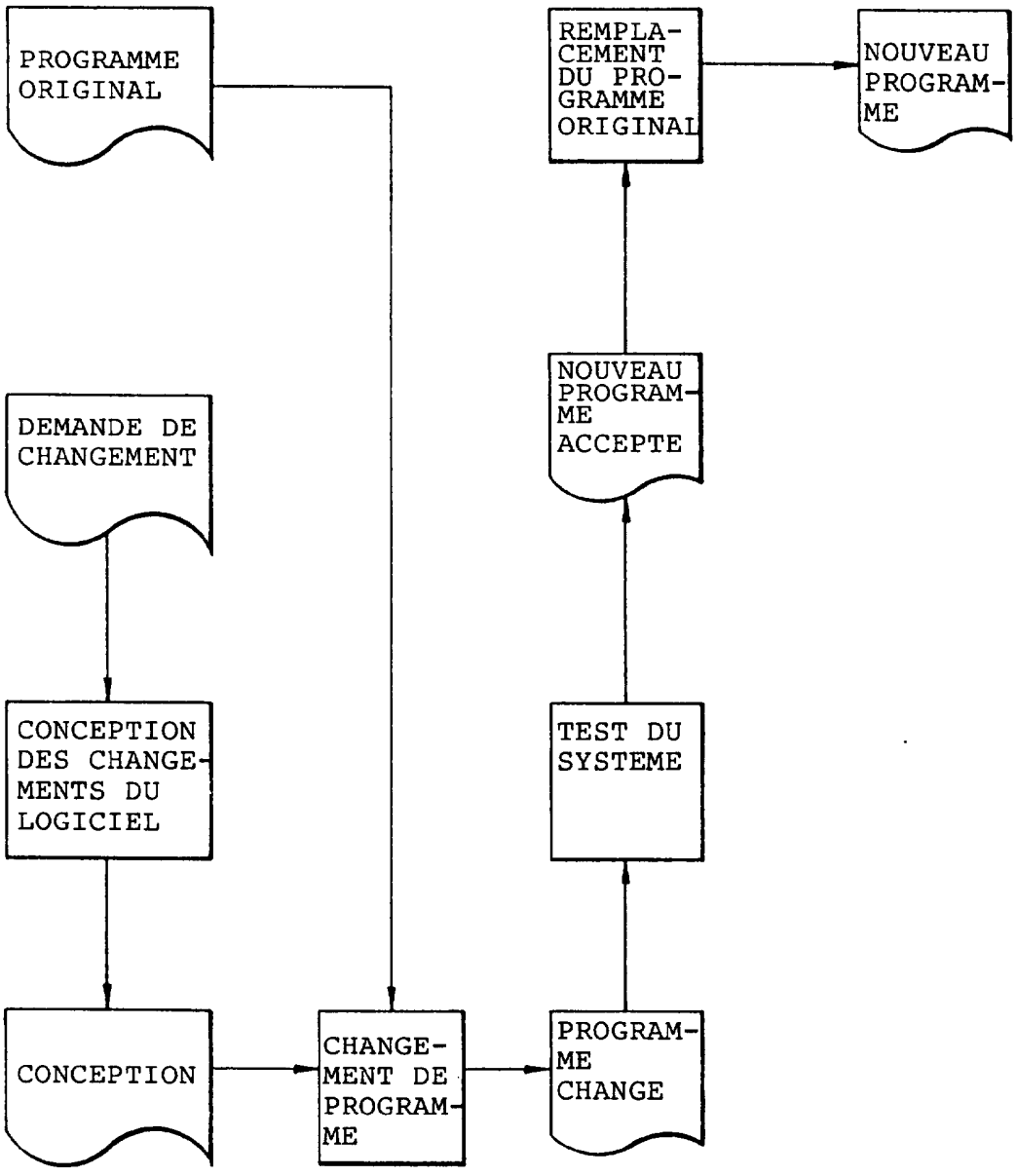

PROGRAMME ORIGINAL
REMPLACEMENT DU PROGRAMME ORIGINAL
NOUVEAU PROGRAMME
DEMANDE DE CHANGEMENT
NOUVEAU PROGRAMME ACCEPTE
CONCEPTION DES CHANGEMENTS DU LOGICIEL
TEST DU SYSTEME
CONCEPTION
CHANGEMENT DE PROGRAMME
PROGRAMME CHANGE

FIG.2

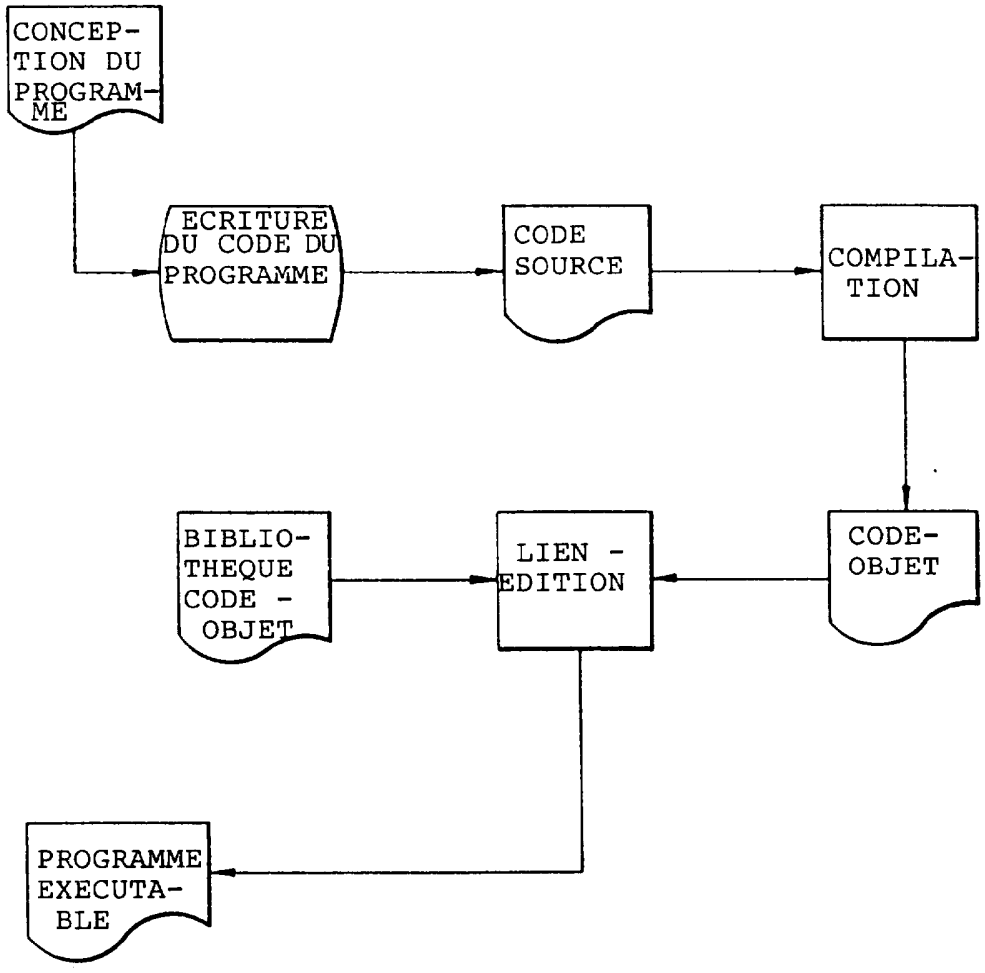

CONCEP-
TION DU
PROGRAM-
ME
ECRITURE
DU CODE DU
PROGRAMME
CODE
SOURCE
COMPILA-
TION
BIBLIO-
THEQUE
CODE -
OBJET
LIEN -
EDITION
CODE-
OBJET
PROGRAMME
EXECUTA-
BLE

FIG.3

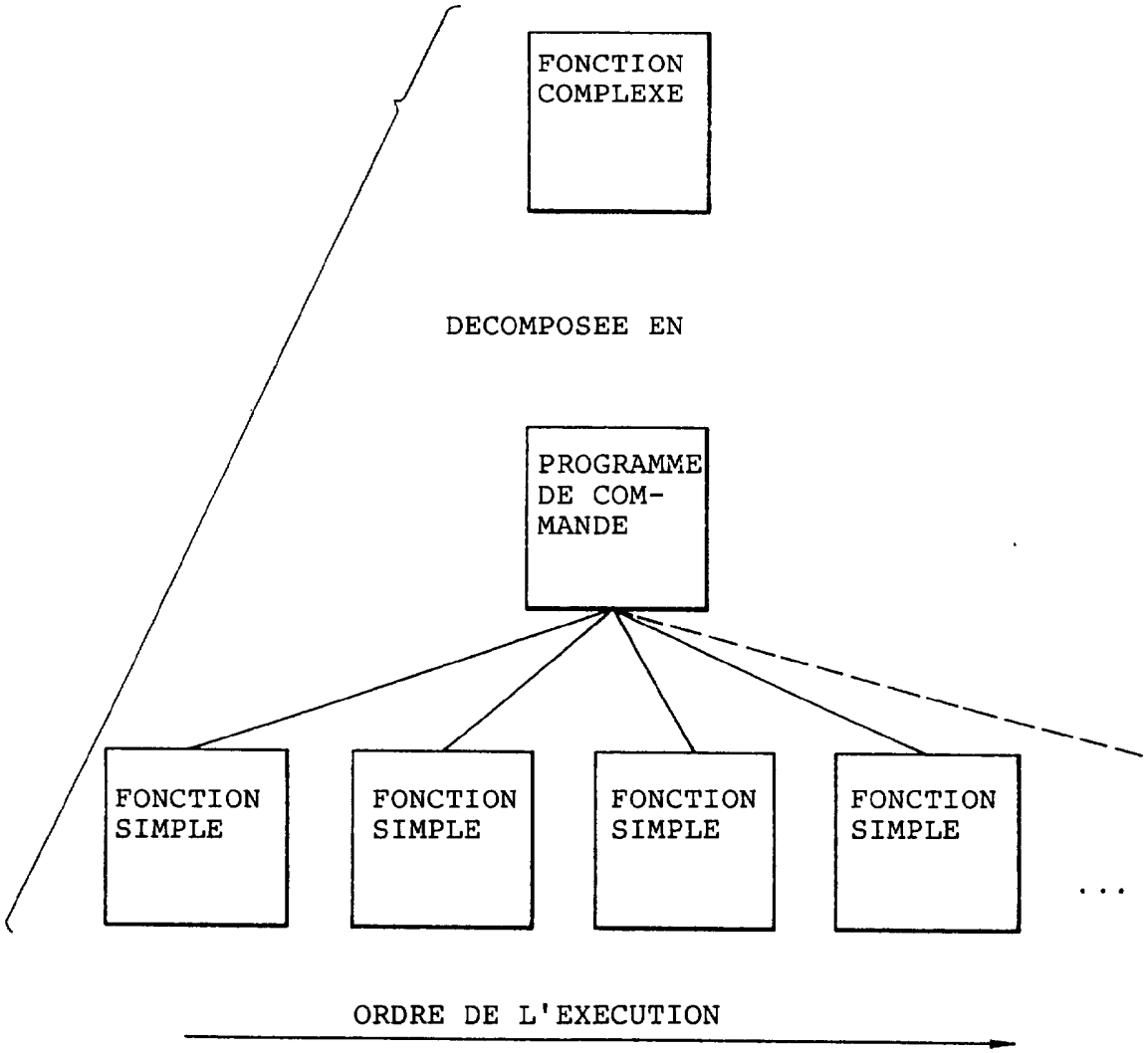

FONCTION COMPLEXE
DECOMPOSEE EN
PROGRAMME DE COM-MANDE
FONCTION SIMPLE
FONCTION SIMPLE
FONCTION SIMPLE
FONCTION SIMPLE
...
ORDRE DE L'EXECUTION

FIG.4

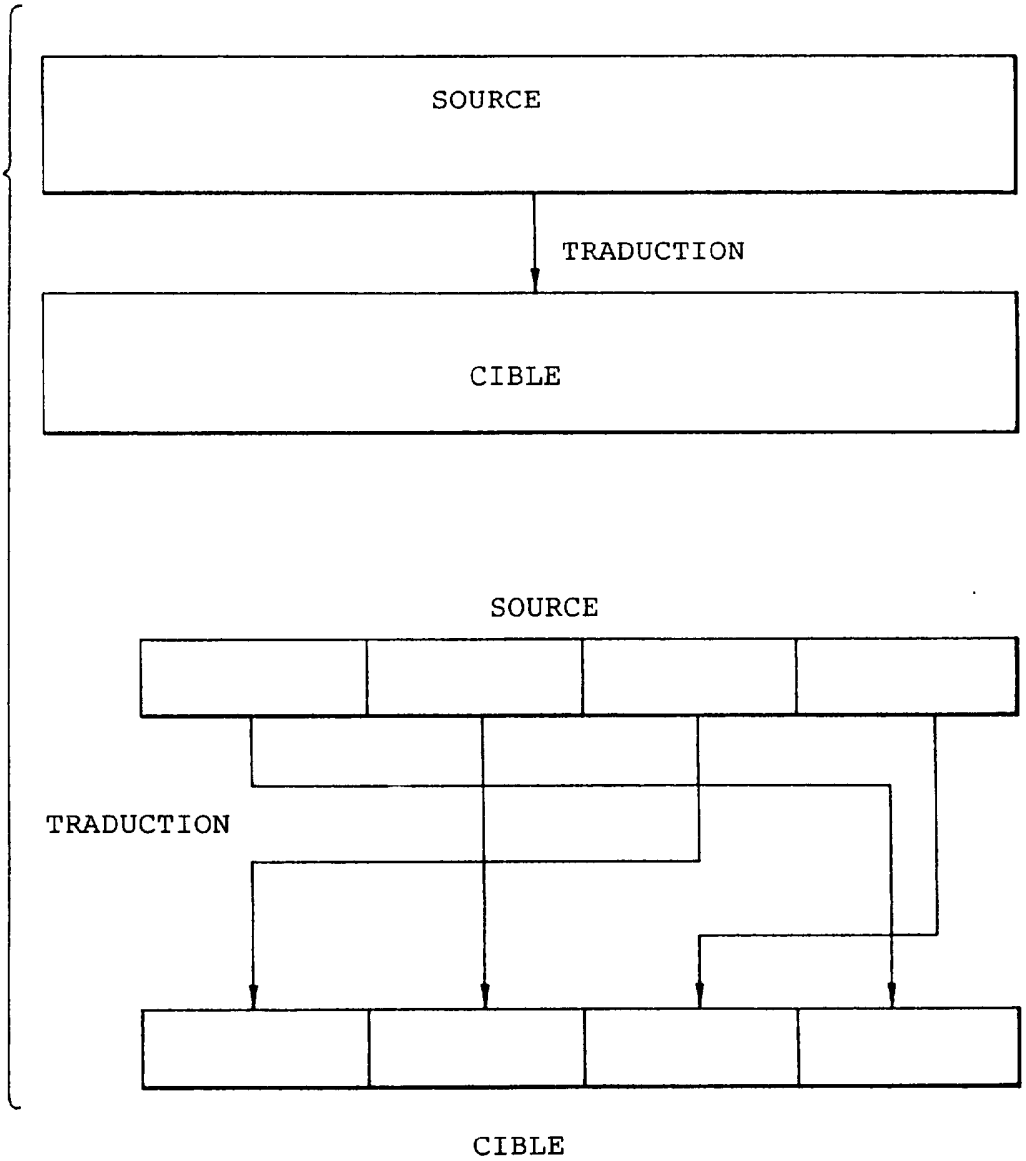


FIG.5

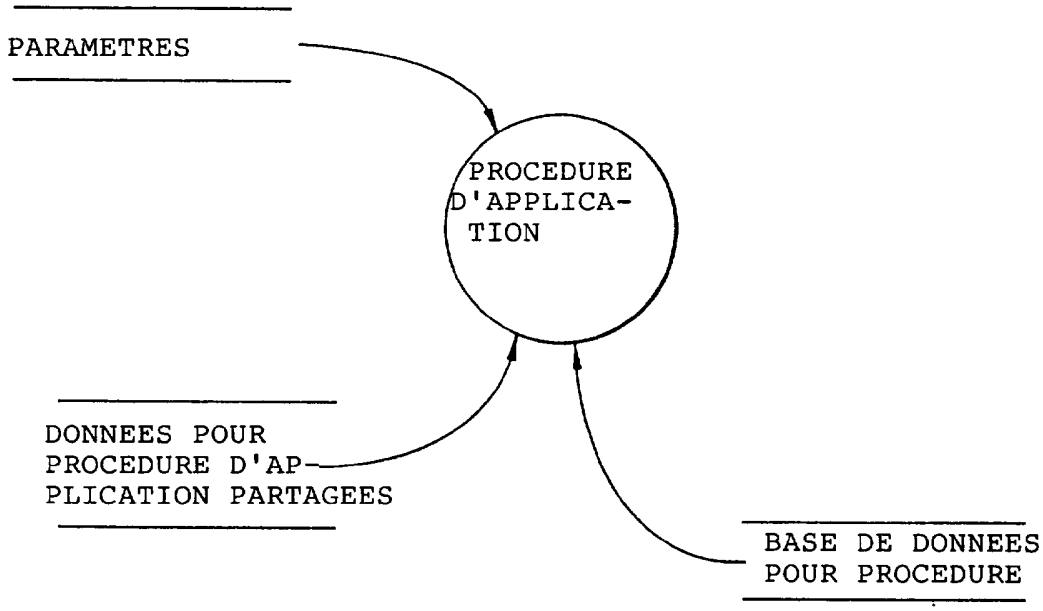
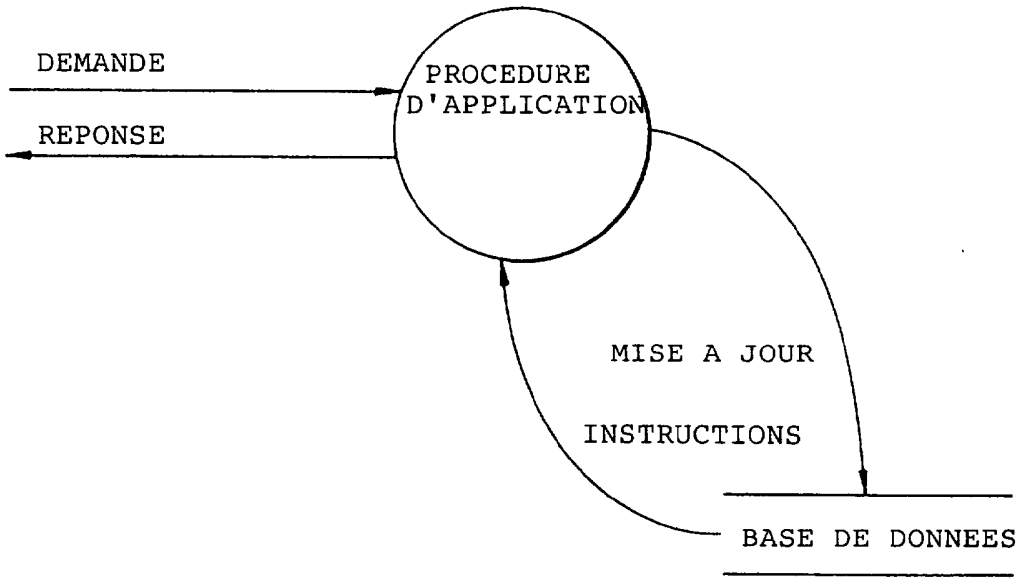


FIG.6

PROCEDURE DE REFERENCE	PARAMETRES

FIG.7


DEMANDE
REPONSE
PROCEDURE
D'APPLICATION
MISE A JOUR
INSTRUCTIONS
BASE DE DONNEES

Numero de la demande nationale

Office européen des brevets

RAPPORT DE RECHERCHE

établi en vertu de l'article 21 § 1 et 2 de la loi belge sur les brevets d'invention du 28 mars 1984

BO 5564
BE 9401092

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.6)
A	US-A-5 175 828 (HALL MICHAEL L ET AL) 29 Décembre 1992 * colonne 2, ligne 13 - ligne 56 *	1-48	G06F9/40
A	GB-A-2 140 942 (HITACHI LTD) 5 Décembre 1984 * page 3, colonne de droite, ligne 94 - page 4, colonne de gauche, ligne 42 *	1-48	
A	IBM TECHNICAL DISCLOSURE BULLETIN, vol. 17, no. 11, Avril 1975 NEW YORK, US, pages 3405-3407, ANONYMOUS 'Rules Driven System Approach. April 1975.' * le document en entier *	1-48	

DOMAINES TECHNIQUES RECHERCHES (Int.Cl.6)

G06F

Date d'achèvement de la recherche	Examinateur
25 Septembre 1995	Brandt, J

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

EPO FORM 1503 03.82 (P04C48)

NNEXE AU RAPPORT DE RECHERCHE
ELATIF A LA DEMANDE DE BREVET BELGE NO.

BO 5564
BE 9401092

La présente annexe indique les membres de la famille de brevets relatifs aux documents brevets cités dans le rapport de recherche visé ci-dessus.
Lesdits membres sont contenus au fichier informatique de l'Office européen des brevets à la date du
Les renseignements fournis sont donnés à titre indicatif et n'engagent pas la responsabilité de l'Office européen des brevets.

25-09-1995

Document brevet cité au rapport de recherche	Date de publication	Membre(s) de la famille de brevet(s)	Date de publication
US-A-5175828	29-12-92	AUCUN	
GB-A-2140942	05-12-84	JP-C- 1854754	07-07-94
		JP-A- 59205605	21-11-84
		GB-A,B 2175112	19-11-86
		US-A- 4683549	28-07-87